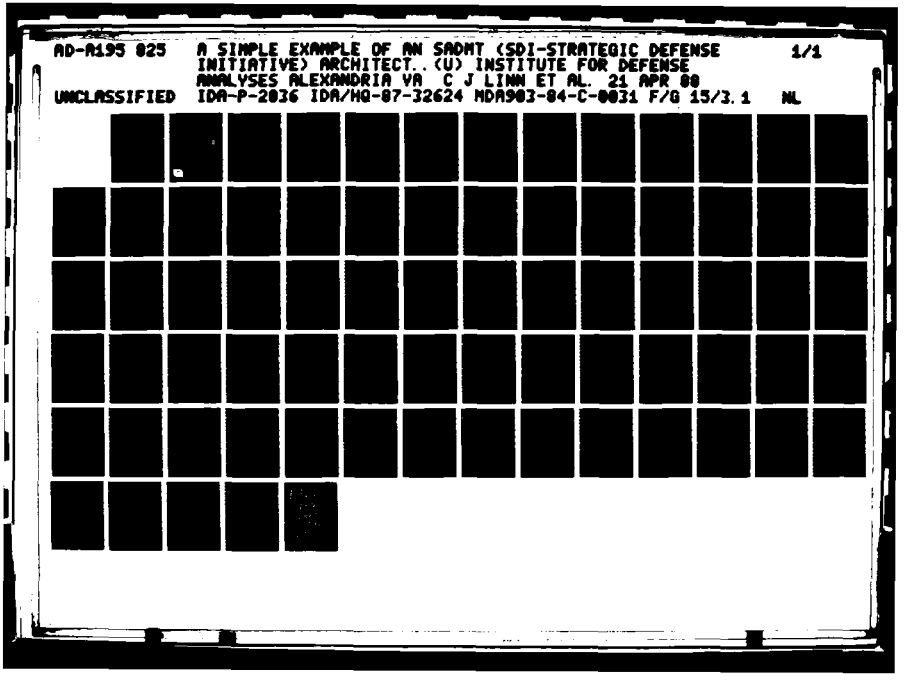
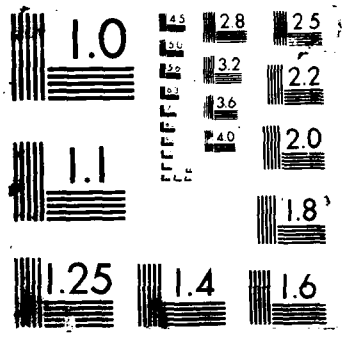

AD-A195 025 A SIMPLE EXAMPLE OF AN SADMT (SDI-STRATEGIC DEFENSE INITIATIVE) ARCHITECT.. (U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA C J LINN ET AL. 21 APR 88
1/1
UNCLASSIFIED IDA-P-2036 IDA/HQ-87-32624 MDA903-84-C-0031 F/G 15/3.1 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

(2)

IDA PAPER P-2036

A SIMPLE EXAMPLE OF AN SADMT ARCHITECTURE SPECIFICATION

VERSION 1.5

Cy D. Ardoin
Stephen H. Edwards
Michael R. Kappel
Cathy Jo Linn
Joseph L. Linn
John Salasin

April 1988

Prepared for
Strategic Defense Initiative Organization (SDIO)

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311

IDA Log No. HQ 87-32624

DEFINITIONS

IDA publishes the following documents to report the results of its work.

Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, or (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Papers

Papers normally address relatively restricted technical or policy issues. They communicate the results of special analyses, interim reports or phases of a task, ad hoc or quick reaction work. Papers are reviewed to ensure that they meet standards similar to those expected of refereed papers in professional journals.

Memorandum Reports

IDA Memorandum Reports are used for the convenience of the sponsors or the analysts to record substantive work done in quick reaction studies and major interactive technical support activities; to make available preliminary and tentative results of analyses or of working group and panel activities; to forward information that is essentially unanalyzed and unevaluated; or to make a record of conferences, meetings, or briefings, or of data developed in the course of an investigation. Review of Memorandum Reports is suited to their content and intended use.

The results of IDA work are also conveyed by briefings and informal memoranda to sponsors and others designated by the sponsors, when appropriate.

The work reported in this document was conducted under contract MDA 903 84 C 0031 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that agency.

This paper has been reviewed by IDA to ensure that it meets high standards of thoroughness, objectivity, and sound analytical methodology and that the conclusions stem from the methodology.

Approved for public release; distribution unlimited.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

AD-A195 225

1a REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release - distribution unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) P-2036	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses 6b OFFICE SYMBOL IDA	7a NAME OF MONITORING ORGANIZATION OUSDA DIMO
6c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311	7b ADDRESS (City, State, and Zip Code) 1801 N. Beauregard Street Alexandria, Virginia 22311
8a NAME OF FUNDING/SPONSORING ORGANIZATION Strategic Defense Initiative Organization 8b OFFICE SYMBOL (If applicable) SDIO	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA 903 84 C 0031
8c ADDRESS (City, State, and Zip Code) SDIO/PI Room 1E149 Pentagon, Washington D.C. 20301-7100	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. T-R5-422 / WORK UNIT ACCESSION NO.
11 TITLE (Include Security Classification) A Simple Example of an SADMT Architecture Specification, Version 1.5 (UNCLASSIFIED)	
12 PERSONAL AUTHOR(S) C. J. Linn, J. Linn, S. H. Edwards, M. R. Kappel, C. D. Ardoin, J. Salasin	
13a TYPE OF REPORT Final 13b TIME COVERED FROM_____ TO _____ 14 DATE OF REPORT (Year, Month, Day) 21 April 1988 15 PAGE COUNT 75	
16 SUPPLEMENTARY NOTATION	
17 COSATI CODES FIELD / GROUP / SUB-GROUP	18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) System Architecture; Dataflow Modeling; Simulation; Program Design Language; Battle Management (BM); Command, Control and Communications (C3).

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This IDA Paper P-2036 presents a simple architecture specification in the SDI Architecture Dataflow Modeling Technique (SADMT). The example code is given in the SADMT Generator (SAGEN) Language. This simple architecture includes (1) an informal description of the architecture, (2) the main program that creates the components of the simulation (3) the specification of the BM/C3 logical processes of the architecture, (4) the specification of the Technology Modules (TMs) of the architecture, and (5) the specification of the BM/C3 and TMs of the threat.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ■ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Dr. Cathy Jo Linn, IDA	22b TELEPHONE (Include Area Code) (703) 824-5520 22c OFFICE SYMBOL IDA/CSED

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

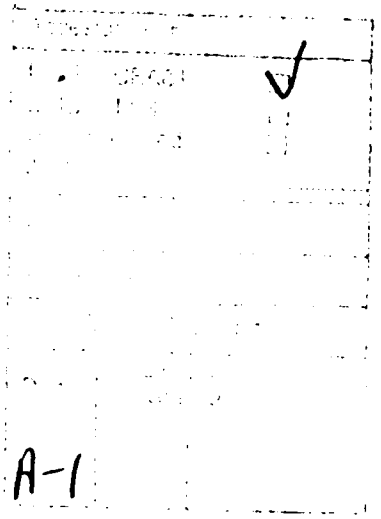
SECURITY CLASSIFICATION OF THIS PAGE

IDA PAPER P-2036

A SIMPLE EXAMPLE OF AN SADMT ARCHITECTURE SPECIFICATION
VERSION 1.5

Cy D. Ardoin
Stephen H. Edwards
Michael R. Kappel
Cathy Jo Linn
Joseph L. Linn
John Salasin

April 1988



IDA

INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
Task T-R5-422

TABLE OF CONTENTS

A Simple Example of an SADMT Architecture Specification

1. Introduction

In this paper, a complete specification of a simple SDI Architecture, SDS0 (pronounced SDS naught), and threat is given. This specification includes:

(1) an informal description of SDS0,

(2) the main program that creates the components of the simulation,

(3) the specification of the BM/C^3 logical processes of the SDS0,

(4) the specification of the Technology Modules (TMs) used by SDS0,

(5) the specification of the BM/C^3 and TMs of the threat.

This example illustrates the manner in which an SDS architecture can be described in SADMT. The entire architecture, BM/C^3 and TMs, is described in SAGEN and was translated by the SAGEN processor into SADMT before being executed.

2. Informal Specification

Components

This sample architecture contains the following components:

Two command posts

Four sensor platforms in geosynchronous orbit

n weapons platforms in low orbit

These components send (through SADMT ports) the following kinds of messages to each other:

(1) Command post to all satellites: the current status.

(2) Sensor platforms to all components: target detection information

(3) Sensor platforms to all components: summaries of target information messages received from other sensor platforms.

The system is always in one of two status levels: war or peace, and the system is initially at peace.

Processing

The following is an outline of the processing that takes place within this system. A command post receives target information continuously from the sensor satellites. It broadcasts a status message (war or peace) to all satellites every ten minutes. When it first receives a message indicating that targets have been detected, it changes status from peace to war and broadcasts that new status immediately. If at war, and no targets are detected for over one hour, the status is changed back to peace.

A sensor scans a designated area an reports all detected targets to all satellites and command posts. In addition, it rebroadcasts a summary of any message it receives from a sensor platform with a higher ID than its own. (The senor platforms are ordered 1 through 4.)

A full sensor message contains the ID of the sensor sending it, the number of targets detected, and a list of information about the targets (including the current position and velocity of each target). A relayed summary message contains only the ID of the sensor platform that is

relaying it, and the number of targets detected.

A weapons platform receives all messages transmitted by the command posts and sensor platforms. It processes the status messages from the command posts, and as long as the status remains at peace it does not care about the contents of any message form the sensors. Nevertheless, after the weapons platform receives a message changing the status to war, it processes all messages that come directly from a sensor platform (*i.e.*, those that contain the complete target information, not the relayed summaries).

For each sensor message processed, the weapons platform calculates the target with the highest probability of kill (P_k). If that P_k is higher that some minimum, it aims and shoots at that target.

The weapon fired is a dumb KKV that travels in the direction it is fired at a constant velocity of 5 km/sec. The weapons platform does not bother to count its remaining KKVs; it keep on playing the game even after it has exhausted its supply.

3. MAIN

The following code is the main program for the SADMT/SF simulation of SDS0. This program creates the initial configuration of platform and starts the simulation.

```
with System_Scheduler,
  Cones_n_Platforms,
  Latitude_n_Longitude,
  Sensor_Cone_Response_TM_pkg,
  Tracker_TM_pkg,
  Platform_Collision_TM_pkg,
  Russian_Missile_Base_Platform_pkg,
  Command_Post_pkg,
  Sensor_Platform_pkg,
  Weapons_Platform_pkg,
  Math,
  Vector_pkg,
  Debug_flags,
  VERDIX;


procedure main_sds0 is
  package PDL_IO renames Cones_n_Platforms.PDL_pkg.PDL_IO;
  package CnP_IP renames Cones_n_Platforms.interface_procs;
  use PDL_IO; use txt_io, int_io;
  use Cones_n_Platforms,
    System_Scheduler,
    Vector_pkg,
    Cones_n_Platforms.eqn_motion_pkg,
    Latitude_n_Longitude,
    Russian_Missile_Base_Platform_pkg,
    Command_Post_pkg,
    Sensor_Platform_pkg,
    Weapons_Platform_pkg,
    Math;
  use PDL_pkg;
  use Command_Post_PARAM_pkg;
  use Sensor_Platform_PARAM_pkg;
  use Weapons_Platform_PARAM_pkg;

  num_sensor_platforms: constant:= 4;

  SECONDS: constant:= PDL_ticks_per_second;
  MINUTES: constant:= 60 * SECONDS;
  HOUR: constant:= 60 * MINUTES;

  missile_base_eom: eqn_motion_type:= new_eqn_motion_rec;
  post1_eom: eqn_motion_type:= new_eqn_motion_rec;
```

```
  post1_param: Command_Post_parameterization_ptr:=
    new Command_Post_parameterization;
  post2_eom: eqn_motion_type:= new_eqn_motion_rec;
  post2_param: Command_Post_parameterization_ptr:=
    new Command_Post_parameterization;
  sensor_eom: eqn_motion_type;
  sensor_param: Sensor_Platform_parameterization_ptr;
  sensor_discr: PDL_string_ptr;
  weapon_eom: eqn_motion_type;
  weapon_param: Weapons_Platform_parameterization_ptr;
  sensor_radius: constant:= Re * 3.875;
  weapon_radius: constant:= Re * 1.125;
  weapon_discr: PDL_string_ptr;
  theta: float;
  platform_pos: vector;

begin

  put_line("The SDS0 simulation:");

  CnP_IP.platforms_cant_collide(Russian_Missile_Base_Platform_designator,
            Russian_Missile_Base_Platform_designator);
  CnP_IP.platforms_cant_collide(Russian_Missile_Base_Platform_designator,
            Command_Post_designator);
  CnP_IP.platforms_cant_collide(Command_Post_designator,
            Command_Post_designator);
  CnP_IP.platforms_cant_collide(Sensor_Platform_designator,
            Sensor_Platform_designator);
  CnP_IP.platforms_cant_collide(Russian_Missile_Base_Platform_designator,
            Weapons_Platform_designator);

  put_line("    Creating Russian Missile Base at " &
    "(55.8 degrees E, 37.9 degrees N).");
  missile_base_eom.position:= location(37.9, 55.8);
  missile_base_eom.delta_t:= max_PDL_duration;
  missile_base_eom.back_ptr_flag:= true;
  missile_base_eom.next_rec:= missile_base_eom;
  Russian_Missile_Base_Platform_CP_pkg.create_platform(
            Russian_Missile_Base_Platform_designator,
            name=> Russian_Missile_Base_Platform_name,
            initial_position=> location(37.9, 55.8),
            eqn_motion=> missile_base_eom);
  put_line("    Creating Command Post 1 at " &
    "(255.0 degrees E, 37.5 degrees N).");
  post1_param.CP_id:= "CPost 1";
  post1_eom.position:= location(37.5, 255.0);
  post1_eom.delta_t:= max_PDL_duration;
  post1_eom.back_ptr_flag:= true;
  post1_eom.next_rec:= post1_eom;
  Command_Post_CP_pkg.create_platform(Command_Post_designator,
            name=> Command_Post_name,
            discr=> PDL_string_ptr'(
                        new string'("1")),
                        param => post1_param,
                        initial_position=> location(37.5,
                        255.0),
                        eqn_motion=> post1_eom);
  put_line("    Creating Command Post 2 at " &
    "(270.0 degrees E, 37.5 degrees N).");
  post2_param.CP_id:= "CPost 2";
  post2_eom.position:= location(37.5, 270.0);
  post2_eom.delta_t:= max_PDL_duration;
  post2_eom.back_ptr_flag:= true;
  post2_eom.next_rec:= post2_eom;
  Command_Post_CP_pkg.create_platform(Command_Post_designator,
            name=> Command_Post_name,
```

```
                discr=> PDL_string_ptr'(
                        new string'("2")),
                        param => post2_param,
                        initial_position=> location(37.5,
                        270.0),
                        eqn_motion=> post2_eom);
  put("Creating ");
  put(num_sensor_platforms,1);
  put(" Sensor Platforms ");
  for ij in 1..num_sensor_platforms loop
    sensor_eom:= new_eqn_motion_rec;
    theta:= float(ij - 1) * 2.0 * PI / float(num_sensor_platforms);
    platform_pos.x:= cos(theta) * sensor_radius;
    platform_pos.y:= sin(theta) * sensor_radius;
    platform_pos.z:= 0.0;
    sensor_eom.position:= platform_pos;
    sensor_eom.delta_t:= max_PDL_duration;
    sensor_eom.back_ptr_flag:= true;
    sensor_eom.next_rec:= sensor_eom;
    sensor_param:= new Sensor_Platform_parameterization;
    sensor_param.SP_id:= "Sensor ";
    put(sensor_param.SP_id(7..7),ij);
    sensor_discr:= new string'(integer'image(ij));
    Sensor_Platform_CP_pkg.create_platform(Sensor_Platform_designator,
                name=> Sensor_Platform_name,
                discr=> sensor_discr,
                param => sensor_param,
                initial_position=> platform_pos,
                eqn_motion=> sensor_eom);
    put('.');
  end loop;
  new_line;
  put("Creating ");
  put(Debug_Flags.num_weapons_platforms,1);
  put(" Weapons Platforms ");
  for ij in 1..Debug_Flags.num_weapons_platforms loop
    weapon_eom:= new_eqn_motion_rec;
    theta:= float(ij - 1) * 2.0 * PI / float(
      Debug_Flags.num_weapons_platforms);
    platform_pos.x:= cos(theta) * weapon_radius;
    platform_pos.y:= sin(theta) * weapon_radius;
    platform_pos.z:= 0.0;
    weapon_eom.position:= platform_pos;
    weapon_eom.delta_t:= max_PDL_duration;
    weapon_eom.back_ptr_flag:= true;
    weapon_eom.next_rec:= weapon_eom;
    weapon_param:= new Weapons_Platform_parameterization;
    weapon_param.WP_id:= "WP    ";
    put(weapon_param.WP_id(3..7),ij);
    weapon_discr:= new string'(integer'image(ij));
    Weapons_Platform_CP_pkg.create_platform(Weapons_Platform_designator,
                param => weapon_param,
                name => Weapons_Platform_name,
                discr => weapon_discr,
                initial_position=> platform_pos,
                eqn_motion=> weapon_eom);
  put('.');
  end loop;
  new_line;
  put_line("simulation begins.....");
  start_simulation(PDL_time_type(HOUR));

end;
```

4. Message and Port types

The message and port types represent the data that is transferred between processes via SADMT ports and the data that is communicated between platforms via SADMT cones. SADMT defines three message types, and these correspond to the three types of outports of the *PIG*. All three types are available to the SADMT user via the *Cones_n_Platforms* package. Nevertheless, three packages which conform to the SAGEN naming conventions are provided. The renamed types are in the following packages:

-------- Renamings of SADMT Packages --------

```
with Cones_n_Platforms;

package Cone_Msg_pkg is

   package WP_EMp renames Cones_n_Platforms.Environ_Msg_pkg;

   subtype Cone_Msg is WP_EMp.Cone_Msg;

   package PD renames WP_EMp.PDcone;

   subtype Cone_Msg_port is PD.T_port;
   subtype Cone_Msg_ipptr is PD.T_ipptr;
   subtype Conse_Msg_opptr is PD.T_opptr;

end Cone_Msg_pkg;
with Cones_n_Platforms;

package Event_Msg_pkg is

   package WP_EMp renames Cones_n_Platforms.Environ_Msg_pkg;

   subtype Event_Msg is WP_EMp.Event_Msg;

   function end_of_eqn_motion return Event_Msg
     renames WP_EMp.end_of_eqn_motion;
   function end_of_lifetime return Event_Msg
     renames WP_EMp.end_of_lifetime;
   function "="(l,r:Event_Msg) return Boolean renames WP_EMp."=";

   package PD renames WP_EMp.PDevent;

   subtype Event_Msg_port is PD.T_port;
   subtype Event_Msg_ipptr is PD.T_ipptr;
   subtype Event_Msg_opptr is PD.T_opptr;

end Event_Msg_pkg;
with Cones_n_Platforms;

package Platform_Msg_pkg is

   package WP_EMp renames Cones_n_Platforms.Environ_Msg_pkg;

   subtype Platform_Msg is WP_EMp.Platform_Msg;

   package PD renames WP_EMp.PDplat;

   subtype Platform_Msg_port is PD.T_port;
   subtype Platform_Msg_ipptr is PD.T_ipptr;
   subtype Platform_Msg_opptr is PD.T_opptr;

end Platform_Msg_pkg;
```

The *boolean* type is used by the *Timer* process of the command post as an alarm signal, and the *time* type is used to set the *Timer* process's alarm clock. The two Ada packages that define the *boolean* and *time* message types and port types are given below

-------- BOOLEAN_PKG --------

```
with PortDefiner_pkg, Cones_n_Platforms;
use Cones_n_Platforms;
package Boolean_pkg is


  type Boolean_ptr is access Boolean;

  Boolean_debug_class: string(1..7):= "Boolean";
  procedure put_msg(m:Boolean; indent:integer:=35);
  package PD is
      new PortDefiner_pkg (
                  Boolean,
                  put_msg,
                  "BOOLEAN",
                  Boolean_debug_class);

  subtype Boolean_port is PD.T_port;
  subtype Boolean_ipptr is PD.T_ipptr;
  subtype Boolean_opptr is PD.T_opptr;

end Boolean_pkg;
package body Boolean_pkg is
  procedure put_msg(m:Boolean;indent:integer:=35) is
    use PDL_pkg.PDL_IO; use TXT_IO;
  begin
    for i in 1..indent loop
      put(' ');
    end loop;
    put("BOOLEAN= ");
    if m then put("TRUE");
    else put("FALSE");
    end if;
    new_line;
  end put_msg;
end Boolean_pkg;
```

-------- TIME_PKG --------

```
with PortDefiner_pkg, Cones_n_Platforms;
use Cones_n_Platforms;
package Time_pkg is

  subtype Time is PDL_pkg.PDL_duration_type;

  type Time_ptr is access Time;

  Time_debug_class: string(1..4):= "Time";
  procedure put_msg(m:Time;indent:integer:=35);
  package PD is
      new PortDefiner_pkg(Time,put_msg,"TIME",Time_debug_class);

  subtype Time_port is PD.T_port;
  subtype Time_opptr is PD.T_opptr;
```

```
  subtype Time_ipptr is PD.T_ipptr;

end Time_pkg;
package body Time_pkg is
  procedure put_msg(m:Time;indent:integer:=35) is
    use PDL_pkg.PDL_IO; use TXT_IO, DURATION_IO;
  begin
    for i in 1..indent loop
      put(' ');
    end loop;
    put("Time= ");
    put(m);
    new_line;
  end put_msg;
end Time_pkg;
```

Next we define an *order* as an enumerated type representing the defcon levels that a ground station can broadcast to the platforms. The enumeration is (*DEFCON1*, *DEFCON2*, *DEFCON3*, *DEFCON4*, *DEFCON5*, *DEFCON6*, *DEFCON7*). Also defined, in a separate package, is the type *order_ptr*. The *ConeDefiner_pkg* package is instantiated with the *order* and *order_ptr* types since *order*s are transmitted via SADMT cones. The definitions of *order* and *order_ptr* are separated to conform with the naming conventions of SAGEN. The packages *Order_pkg* and *Order_ptr_pkg* are given below:

-------- ORDER_PKG --------

```
with PortDefiner_pkg,
   Cones_n_Platforms;
package Order_pkg is
  use Cones_n_Platforms;

  type Defense_status is (DEFCON1, DEFCON2, DEFCON3, DEFCON4,
         DEFCON5, DEFCON6, DEFCON7);
  type Order is
    record
      Initiator: string (1..7):= "------";
      order: Defense_status;
    end record;

  Order_debug_class: string(1..5):= "ORDER";
  procedure put_msg(
            m:Order;
            indent:integer:=20);
  package PD is new PortDefiner_pkg(Order,put_msg,Order_debug_class);
  package Defense_status_IO is
     newPDL_pkg.PDL_IO.TXT_IO.ENUMERATION_IO(Defense_status);
  subtype Order_port is PD.T_port;
  subtype Order_ipptr is PD.T_ipptr;
  subtype Order_opptr is PD.T_opptr;

end Order_pkg;

package body Order_pkg is
  procedure put_msg(
            m: Order;
            indent:integer:=20) is
    use PDL_pkg.PDL_IO; use TXT_IO;
    use Defense_status_IO;
  begin
    for i in 1..indent loop
```

```
      put(' ');
    end loop;
    put("Order: initiator=");
    put_line(m.Initiator);
    for i in 1..indent+3 loop
      put(' ');
    end loop;
    put("status=");
    Defense_status_IO.put(m.order);
    new_line;
  end put_msg;
end Order_pkg;
```

-------- ORDER_PTR_PKG --------

```
with Cones_n_Platforms,
  Order_pkg;
package Order_ptr_pkg is
  use Cones_n_Platforms;
  use Order_pkg;

  type Order_ptr is access Order;
  Command_Transmission: constant cone_designator_type

  package CD is new interface_procs.ConeDefiner_pkg(Order,Order_ptr);

  package CAST is new Casting_Functions(Order, Order_ptr);
  function cast_magic_ptr_into_order_ptr(ptr: in PDL_magic_ptr)
    return Order_ptr renames CAST.CAST_magic_ptr_INTO_T_ptr;
  function cast_order_ptr_into_magic_ptr(ptr: in Order_ptr)
    return PDL_magic_ptr renames CAST.CAST_T_ptr_INTO_magic_ptr;
end Order_ptr_pkg;
```

The *target* type is a record with latitude and longitude entries used by the Russian missile to communicate target locations between its subprocesses. The packages *Target_pkg* and *latitude_n_longitude* are given below:

-------- TARGET_PKG --------

```
with PortDefiner_pkg;

package Target_pkg is
  type Target is record
    latitude: float;
    longitude: float;
  end record;

  Target_debug_class: string(1..6):= "Target";
  procedure put_msg(
            m:Target;
            indent:integer:=20);
  package PD is
      new PortDefiner_pkg(
                Target,
                put_msg,
                "TARGET",
                Target_debug_class);
```

```
  subtype Target_port is PD.T_port;
  subtype Target_ipptr is PD.T_ipptr;
  subtype Target_sipptr is PD.T_sipptr;
  subtype Target_opptr is PD.T_opptr;
  subtype Target_sopptr is PD.T_sopptr;
end Target_pkg;

with PDL_pkg;
package body Target_pkg is
  procedure put_msg(
            m:Target;
            indent:integer:=20) is
    use PDL_pkg.PDL_IO; use TXT_IO, FLT_IO;
  begin
    for i in 1..indent loop
      put(' ');
    end loop;
    put("Target: Latitude=");
    put(m.latitude);
    put(" Longitude=");
    put(m.longitude);
    new_line;
  end put_msg;
end Target_pkg;
```

-------- LATITUDE_N_LONGITUDE --------

```
with Cones_n_Platforms,
  Vector_pkg,
  Math;

package latitude_n_longitude is
  use Vector_pkg;

  Re: constant:= 6.378145E3 *
    Cones_n_Platforms.PDL_Junits_per_kilometer;

  function location (latitude, longitude: in float) return vector;
end latitude_n_longitude;

package body latitude_n_longitude is
  use Math;

  function location (latitude, longitude: in float) return vector is
    pos: vector;
    rad_lat: float:= latitude  * PI / 180.0;
    rad_long: float:= longitude * PI / 180.0;
  begin
    pos.x:= Re * cos(rad_lat) * cos(rad_long);
    pos.y:= Re * cos(rad_lat) * sin(rad_long);
    pos.z:= Re * sin(rad_lat);
    return pos;
  end location;

end latitude_n_longitude;
```

The *sense_req* type is identical to the *cone_type* defined by SADMT. Thus, *sense_req* contains 4 fields, a *source_point*, and *indicator_point*, a *half_angle*, and a *blackout_radius*. This message type is used to specify the conical scope of a sensor scan. Once again the access type *sense_req_ptr* is separate from the base type *sense_req_ptr*. This is done to conform with the naming conventions of SAGEN. The packages *Sense_Req_pkg* and *Sense_Req_ptr_pkg* are given below:

-------- SENSOR_REQ_PKG --------

```
with PortDefiner_pkg,
  Vector_pkg,
  PDL_pkg,
  Cones_n_Platforms;

package Sense_Req_pkg is

  subtype Sense_Req is Cones_n_Platforms.cone_type;

  Sense_Req_debug_class: string(1..13):= "Sense_Request";
  procedure put_msg(
            m:Sense_Req;
            indent:integer:=20);
  package PD is
      new PortDefiner_pkg(
                Sense_Req,
                put_msg,
                "Sense_Request",
                Sense_Req_debug_class);

  subtype Sense_Req_port is PD.T_port;
  subtype Sense_Req_ipptr is PD.T_ipptr;
  subtype Sense_Req_sipptr is PD.T_sipptr;
  subtype Sense_Req_opptr is PD.T_opptr;
  subtype Sense_Req_sopptr is PD.T_sopptr;
end Sense_Req_pkg;

package body Sense_Req_pkg is
  procedure put_msg(
            m:Sense_Req;
            indent:integer:=20) is
    use PDL_pkg.PDL_IO; use TXT_IO, FLT_IO;
    use Vector_pkg;
  begin
    for i in 1..indent loop
      put(' ');
    end loop;
    put("Sense Request: inidicator_pt=");
    put_vector(m.indicator_point,0);
    put(", source_pt=");
    put_vector(m.source_point,0);
    put_line(",");
    for i in 1..indent loop
      put(' ');
    end loop;
    put("Half angle=");
    put(m.half_angle);
    put(", Blackout Radius=");
    put(m.blackout_radius);
    new_line;
  end put_msg;
end Sense_Req_pkg;
```

-------- SENSOR_REQ_PTR_PKG --------

```
with Cones_n_Platforms,
   Sense_Req_pkg;

package Sense_Req_ptr_pkg is
  use Cones_n_Platforms;
  use Sense_Req_pkg;

  type Sense_Req_ptr is access Sense_Req;

  package CD is
      new interface_procs.ConeDefiner_pkg(Sense_Req,Sense_Req_ptr);

end Sense_Req_ptr_pkg;
```

The *Sensor_Cone_pkg* package defines two *cone_designator_types*. The *Sensor_Cone* is used to distinguish a sensor's radar, and a *Sensor_Cone_Reflection* is used to distinguish a radar echo. The package *Sensor_Cone_pkg* is given below:

-------- SENSOR_CONE_PKG --------

```
with Cones_n_Platforms;

package Sensor_Cone_pkg is
  use Cones_n_Platforms;

  Sensor_Cone: constant cone_designator_type
  Sensor_Cone_Reflection: constant cone_designator_type

end Sensor_Cone_pkg;
```

The *Platform_Data_Req_pkg* defines an integer used as a signal to request information about the current state of the platform from the *Platform_Data_TM*. When the TM receives the signal it computes the physical location, speed, velocity, mass, and so on, then this data is sent over the output port in the form of a *Platform_Data* type. These two message types are given below:

-------- PLATFORM_DATA_REQ_PKG --------

```
with PortDefiner_pkg;

package Platform_Data_Req_pkg is
  type Platform_Data_Req is new integer;
  type Platform_Data_Req_type is access Platform_Data_Req;

  Platform_Data_Req_debug_class: string(1..21):= "Platform_data_request";
  procedure put_msg(
            m:Platform_Data_Req;
            indent:integer:=20);
  package PD is
      new PortDefiner_pkg(
                Platform_Data_Req,
                put_msg,
                Platform_Data_Req_debug_class);

  subtype Platform_Data_Req_port is PD.T_port;
  subtype Platform_Data_Req_ipptr is PD.T_ipptr;
  subtype Platform_Data_Req_sipptr is PD.T_sipptr;
  subtype Platform_Data_Req_opptr is PD.T_opptr;
  subtype Platform_Data_Req_sopptr is PD.T_sopptr;
end Platform_Data_Req_pkg;

with PDL_pkg;
package body Platform_Data_Req_pkg is
  procedure put_msg(
            m:Platform_Data_Req;
            indent:integer:=20) is
    use PDL_pkg.PDL_IO; use TXT_IO, INT_IO;
  begin
    for i in 1..indent loop
      put(' ');
    end loop;
    put("Platform_Data_Req=");
```

```
    put(integer(m),1);
    new_line;
  end put_msg;
end Platform_Data_Req_pkg;
```

-------- PLATFORM_DATA_PKG --------

```
with PortDefiner_pkg,
  Cones_n_Platforms,
  Vector_pkg;

package Platform_Data_pkg is
  use Cones_n_Platforms,
    Vector_pkg;

  type Platform_Data is record
    designator: platform_designator_type;
    mass: float;
    eqn_motion: eqn_motion_type;
    current_eqn_motion_segment: eqn_motion_type;
    when_arrived_this_segment: float;
    when_leaving_this_segment: float;
    position: vector;
    speed: float;
    velocity: vector;
  end record;
  type Platform_Data_type is access Platform_Data;

  Platform_Data_debug_class: string(1..13):= "Platform_data";
  procedure put_msg(
            m:Platform_Data;
            indent:integer:=20);
  package PD is
      new PortDefiner_pkg(
                Platform_Data,
                put_msg,
                Platform_Data_debug_class);

  subtype Platform_Data_port is PD.T_port;
  subtype Platform_Data_ipptr is PD.T_ipptr;
  subtype Platform_Data_opptr is PD.T_opptr;
end Platform_Data_pkg;

package body Platform_Data_pkg is
  procedure put_msg(
            m:Platform_Data;
            indent:integer:=20) is
      use PDL_pkg.PDL_IO; use TXT_IO, FLT_IO, DURATION_IO, INT_IO;
    use Vector_pkg;
    procedure put_indent(offset:integer) is
    begin
      new_line;
      for i in 1..offset loop
        put(' ');
      end loop;
    end put_indent;
    procedure put_eqn_mot(
                m:eqn_motion_type;
                indent:integer) is
      p: eqn_motion_type:=m;
      i: integer;
```

```
      back_ptr: boolean:= false;
    begin
      for j in 1..indent loop
        put(' ');
      end loop;
      i:= 1;
      put(i,1);
      while p /= null loop
        put_indent(indent+3);
        put("Position=");
        put_vector(m.position,0);
        put_indent(indent+3);
        put("Delta time=");
        put(m.delta_t,1);
        back_ptr:=true;
        exit when p.back_ptr_flag;
        back_ptr:= false;
        p:= p.next_rec;
        exit when p = null;
        put_indent(indent);
        i:= i+1;
        put(i,1);
      end loop;
      put_indent(indent);
      if back_ptr then
        put_line("***Back_ptr");
      else
        put_line("***NULL");
      end if;
    end put_eqn_mot;
  begin
    for i in 1..indent loop
      put(' ');
    end loop;
    put("Platform Data: ");
    put_indent(indent+3);
    put("Designator=");
    put(m.designator.all);
    put_indent(indent+3);
    put("Mass=");
    put(m.mass);
    put_indent(indent+3);
    put_line("Equation of Motion:");
    put_eqn_mot(m.eqn_motion,indent+6);
    put_indent(indent+3);
    put_line("Current Equation of Motion Segment:");
    put_eqn_mot(m.current_eqn_motion_segment,indent+6);
    put_indent(indent+3);
    put("Arrived in Segment=");
    put(m.when_arrived_this_segment,1);
    put_indent(indent+3);
    put("Leaving Segment=");
    put(m.when_leaving_this_segment,1);
    put_indent(indent+3);
    put("Position=");
    put_vector(m.position,0);
    put_indent(indent+3);
    put("Speed=");
    put(m.speed);
    put_indent(indent+3);
    put("Velocity=");
    put_vector(m.velocity,0);
    new_line;
  end put_msg;
end Platform_Data_pkg;
```

The *Track_Data_pkg* package contains two basic data types. (1) *Sensor_ID* is an integer subtype for identifying sensor satellites. (2) *Track_Data* is a record consisting of a *Sensor_ID*, the number of targets, and a pointer to a linked list of *Track_Data_rec*s. This data type is transmitted between sensor platforms; therefore, an access type and *ConeDefiner_pkg* must also be defined. The access type *Track_Data_ptr* and *ConeDefiner_pkg* are defined in the package *Track_Data_ptr_pkg*. The *Track_Data_rec* type is defined in the package *Track_Data_rec_pkg*. A *Track_Data_rec* contains information about a single platform. It contains fields for the position and the velocity of a platform along with a pointer to form a linked list. The access type, *Track_Data_rec_ptr* is a pointer to a list of *Track_Data_rec*. The three package are given below:

-------- TRACK_DATA_PKG --------

```
with PortDefiner_pkg,
  Track_Data_rec_ptr_pkg;

package Track_Data_pkg is
  use Track_Data_rec_ptr_pkg;

  subtype Sensor_ID is integer;
  type Track_Data is record
    initiator: Sensor_ID;
    number_of_targets: integer;
    track_list: Track_Data_rec_ptr;
  end record;

  Track_Data_debug_class: string(1..10):= "Track_Data";
  procedure put_msg(
            m:Track_Data;
            indent:integer:=20);
  package PD is
      new PortDefiner_pkg(
                Track_Data,
                put_msg,
                "TRACK_DATA",
                Track_Data_debug_class);

  subtype Track_Data_port is PD.T_port;
  subtype Track_Data_ipptr is PD.T_ipptr;
  subtype Track_Data_sipptr is PD.T_sipptr;
  subtype Track_Data_opptr is PD.T_opptr;
  subtype Track_Data_sopptr is PD.T_sopptr;

end Track_Data_pkg;
with PDL_pkg;
package body Track_Data_pkg is
  procedure put_msg(
            m:Track_Data;
            indent:integer:=20) is
    use PDL_pkg.PDL_IO; use TXT_IO, INT_IO;
  begin
    for i in 1..indent loop
      put(' ');
    end loop;
    put("Track Data: initiator=");
    put(integer(m.initiator),1);
    put(" num_of_targets=");
    put(m.number_of_targets,1);
    new_line;
    Track_Data_rec_ptr_pkg.put_msg(m.track_list,indent+5);
  end;
end Track_Data_pkg;
```

-------- TRACK_DATA_PTR_PKG --------

```
with Track_Data_rec_ptr_pkg,
   Track_Data_pkg,
   Cones_n_Platforms;

package Track_Data_ptr_pkg is
  use Track_Data_rec_ptr_pkg, Track_Data_pkg, Cones_n_Platforms;

  type Track_Data_ptr is access Track_Data;
  Sensor_Data_Transmission: constant cone_designator_type

  package CD is
     new interface_procs.ConeDefiner_pkg(
                          Track_Data,
                          Track_Data_ptr);

  package CAST is new Casting_Functions(Track_Data, Track_Data_ptr);
  function cast_magic_ptr_into_track_data_ptr(ptr: in PDL_magic_ptr)
     return Track_Data_ptr renames CAST.CAST_magic_ptr_INTO_T_ptr;
  function cast_track_data_ptr_into_magic_ptr(ptr: in Track_Data_ptr)
     return PDL_magic_ptr renames CAST.CAST_T_ptr_INTO_magic_ptr;

end Track_Data_ptr_pkg;
```

-------- TRACK_DATA_REC_PKG --------

```
with PortDefiner_pkg,
   Vector_pkg,
   Cones_n_Platforms;

package Track_Data_rec_ptr_pkg is
  use Vector_pkg, Cones_n_Platforms;

  type Track_Data_rec;
  type Track_Data_rec_ptr is access Track_Data_rec;
  type Track_Data_rec is record
    position: point_type;
    velocity: point_type;
    next_rec: Track_Data_rec_ptr;
  end record;

  Track_Data_rec_ptr_debug_class: string(1..18):= "Track_Data_Pointer";
  procedure put_msg(
           m:Track_Data_rec_ptr;
           indent:integer:=20);
  package PD is
     new PortDefiner_pkg(
               Track_Data_rec_ptr,
               put_msg,
               "TRACK_DATA_POINTER",
               Track_Data_rec_ptr_debug_class);
  package CD is
     new interface_procs.ConeDefiner_pkg(
                          Track_Data_rec,
                          Track_Data_rec_ptr);

  subtype Track_Data_rec_ptr_port is PD.T_port;
  subtype Track_Data_rec_ptr_ipptr is PD.T_ipptr;
  subtype Track_Data_rec_ptr_sipptr is PD.T_sipptr;
  subtype Track_Data_rec_ptr_opptr is PD.T_opptr;
  subtype Track_Data_rec_ptr_sopptr is PD.T_sopptr;
```

```
  package CAST is
      new Casting_Functions(Track_Data_rec, Track_Data_rec_ptr);
  function cast_magic_ptr_into_track_data_rec_ptr(ptr: in PDL_magic_ptr)
      return Track_Data_rec_ptr
      renames CAST.CAST_magic_ptr_INTO_T_ptr;
  function cast_track_data_rec_ptr_into_magic_ptr(
    ptr: in
    Track_Data_rec_ptr)
      return PDL_magic_ptr renames CAST.CAST_T_ptr_INTO_magic_ptr;

  function new_track_data_rec return Track_Data_rec_ptr;
  procedure free_track_data_rec(r: in out Track_Data_rec_ptr);
  procedure free_track_data_list(r: in out Track_Data_rec_ptr);
end Track_Data_rec_ptr_pkg;

package body Track_Data_rec_ptr_pkg is

  global_store: Track_Data_rec_ptr:= null;

  procedure put_msg(
            m:Track_Data_rec_ptr;
            indent:integer:=20) is
    use Cones_n_Platforms.PDL_pkg.PDL_IO; use TXT_IO, INT_IO;
    use Vector_pkg;
    p: Track_Data_rec_ptr:=m;
    i: integer;
  begin
    for j in 1..indent loop
      put(' ');
    end loop;
    put_line("Track_Data_rec_ptr.data=");
    i:= 1;
    while p /= null loop
      for j in 1..indent+3 loop
        put(' ');
      end loop;
      put(i); put(": position/velocity=");
      put_vector(m.position,0); put('/');
      put_vector(m.velocity,0);
      new_line;
      p:= p.next_rec;
    end loop;
    for j in 1..indent loop
      put(' ');
    end loop;
    put_line("***NULL end track_data_rec_ptr");
  end put_msg;

  function new_track_data_rec return Track_Data_rec_ptr is
    new_rec: Track_Data_rec_ptr:= null;
  begin
    if global_store = null then
      new_rec:= new Track_Data_rec;
    else
      new_rec:= global_store;
      global_store:= global_store.next_rec;
    end if;
    new_rec.next_rec:= null;
    return new_rec;
  end new_track_data_rec;

  procedure free_track_data_rec(r: in out Track_Data_rec_ptr) is
  begin
    if r /= null then
      r.next_rec:= global_store;
      global_store:= r;
```

```
      r:= null;
    end if;
    return;
  end free_track_data_rec;

  procedure free_track_data_list(r: in out Track_Data_rec_ptr) is
    follower: Track_Data_rec_ptr:= r;
  begin
    if r /= null then
      while follower.next_rec /= null loop
        follower:= follower.next_rec;
      end loop;
      follower.next_rec:= global_store;
      global_store:= r;
      r:= null;
    end if;
    return;
  end free_track_data_list;

end Track_data_rec_ptr_pkg;
```

The next package does not define message types or port types; however, it is useful when printing the value of spatial vectors. The package *Vector_IO* defines the procedure *put*. *Put* outputs a vector as the latitude, longitude, and height above the surface of the earth. The package is given below:

-------- VECTOR IO --------

```
with Vector_pkg;

package Vector_IO is
  use Vector_pkg;

  procedure put(
          v: vector;
          fore: integer:= 4;
          aft: integer:= 3;
          exp: integer:= 0);

end Vector_IO;
with PDL_pkg,
  Latitude_n_Longitude,
  Cones_n_Platforms,
  Math;

package body Vector_IO is
  use PDL_pkg.PDL_IO.TXT_IO,
    PDL_pkg.PDL_IO.FLT_IO,
    Math;
  min_len: constant:= 0.0001 *
    Latitude_n_Longitude.Re;

  procedure put(
          v: vector;
          fore: integer:= 4;
          aft: integer:= 3;
          exp: integer:= 0) is
    lat: float;
    long: float;
    height: float;
    temp: float;
  begin
```

```
    if length(v) < min_len then
      put("(Center of the Earth)");
      return;
    end if;
    lat:= arcsin(v.z/length(v));
    temp:= sqrt(v.x*v.x + v.y*v.y);
    if temp = 0.0 then
      long:= 0.0;
    else
      long:= arcsin(v.y/temp);
    end if;
    height:= length(v) - Latitude_n_Longitude.Re;
    lat:= lat  * 180.0 / PI;
    long:= long * 180.0 / PI;
    if v.x < 0.0 then
        long:= 180.0 - long;
    elsif v.y < 0.0 then
      long:= long + 360.0;
    end if;
    height:= height / Cones_n_Platforms.PDL_Junits_per_kilometer;
    put("(lat = ");
    put(lat,fore,aft,exp);
    put(" deg.,long= ");
    put(long,fore,aft,exp);
    put(" deg.,hgt = ");
    put(height,fore,aft,exp);
    put(" km.)");
  exception
    when others =>
      put_line("An untrapped exception occurred in" &
        " Vector_IO.put.");
      raise;
  end;

end Vector_IO;
```

5. SDS0 BM/C^3

The following SAGEN code defines the BM/C^3 processes of SDS0. Figure 5-1 through Figure 5-4 illustrate the SADMT process architecture of the system. Figure 5-1 through Figure 5-3 show the architectures of a *Command_Post*, *Weapon_Platform*, and *Sensor_Platform*, respectively. Figure 5-4 shows the next-level decomposition of the *Sensor_Platform_Processing* process form the *Sensor_Platform* platform.

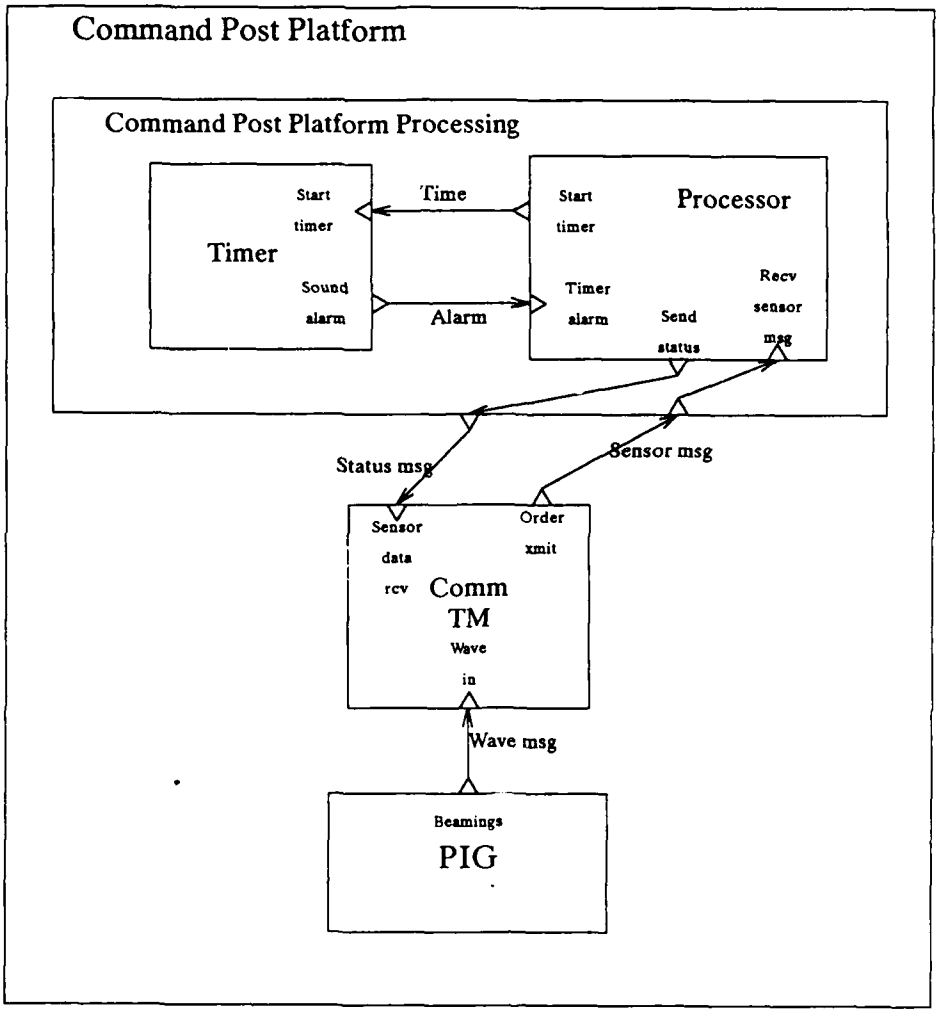


Figure 5-1. Command Post Architecture.

-------- COMMAND POST --------

```
-- ------------------------------------------
--          Command_Post Platform Description
-- ------------------------------------------

-- DESCRIPTION
--
--   The Command Post (CP) enables the system (by sending an "at war"
--   message) once it hears about any targets. It disables the system
--   (by sending an "at peace" message) after hearing about zero
--   targets for more than an hour.
--   The CP sends one of these messages to all satellites every 10 seconds.
--   On a peace-to-war transition, a message is sent immediately.


-- PLATFORM SPECIFICATION

$platform Command_Post:= Command_Post is

  $parameter CP_id: string(1..7):= ("--CP--");

  $subprocess Command_Post_Processing:= (CP_id),
        Ground_Station_Communication_TM:= (CP_id);
        -- The PIG is a predefined subprocess of every platform

  $subdata  Order, Track_data, Cone_msg;

$end;

-- PORT LINKAGES

with Sensor_Cone_Response_TM_pkg;
$links Command_Post is

$begin

   -- Exclude the Radar Return TM
   exclude_dyn_module(MYSELF,
     Sensor_Cone_Response_TM_pkg.Sensor_Cone_Response_TM_designator);

   -- Connect Command_Post to Ground_Station_Communication_TM
   internal_link (Command_Post_Processing.Send_status_msg,
          Ground_Station_Communication_TM.Order_xmit);
   internal_link (Ground_Station_Communication_TM.Sensor_data_rcv,
          Command_Post_Processing.Recv_sensor_msg);
   -- Connect PIG to Ground_Station_Communication_TM
   internal_link (PIG.Beamings,
          Ground_Station_Communication_TM.Cone_in);
$end;


-- PROCESS SEMANTICS
-- The semantics of this platform are defined within its subprocesses.
-- Older versions of Sagen required the $task, this one doesn't

-- ------------------------------------------
--          Command_Post_Processing Process
-- ------------------------------------------

-- PROCESS SPECIFICATION

$process Command_Post_Processing is

  $parameter CP_id: string(1..7):= ("--CP--");
```

```
  $inport   Recv_sensor_msg: Track_data;

  $outport  Send_status_msg: Order;

  $subprocess Processor:= (CP_id),
        Timer:= (CP_id);

  $subdata  Time, Boolean;

$end;

— PORT LINKAGES

$links Command_Post_Processing is

$begin

  — Connect Processor to parent (Command_Post_Processing)
  inherited_link (Recv_sensor_msg,
          Processor.Recv_sensor_msg);
  inherited_link (Processor.Send_status_msg,
          Send_status_msg);
  — Connect Processor to Timer
  internal_link (Processor.Start_timer,
          Timer.Start_timer);
  internal_link (Timer.Sound_alarm,
          Processor.Timer_alarm);

$end;

— TASK SEMANTICS
— The semantics are defined in the subprocesses Processor and Timer.
— Older versions of Sagen required the $task, this one doesn't

————————————————————————————————
—              Processor Process
————————————————————————————————

— PROCESS SPECIFICATION

$process Processor is

  $parameter CP_id: string(1..7):= ("--CP---");

  $inport   Recv_sensor_msg: Track_data,
        Timer_alarm: Boolean;

  $outport  Send_status_msg: Order,
        Start_timer: Time;

$end;

— PORT LINKAGES
— There are no port linkages within this process
— Older versions of Sagen required the $link, this one doesn't

— TASK SEMANTICS

$task Processor is

  Status: Order;
  Last_target_time: PDL_time_type;
  Targets_detected: Boolean;
  Sensor_info: Track_data;

  sec: constant:= PDL_ticks_per_second;
```

```
min: constant:= 60 * PDL_ticks_per_second;

$begin

  -- Start at peace
  Status.Initiator:= CP_id;
  Status.Order:= DEFCON7;

  -- Send status to all satellites
  Emit (Send_status_msg, Status);

  -- *** DEMO MSGS ***
  if Current_debug_level > 20 then
    Put (CP_id);
    Put (" broadcast status ");
    if Status.Order = DEFCON1 then
      Put ("war.");
    elsif Status.Order = DEFCON7 then
      Put ("peace.");
    end if;
    Put (" T=");
    Put (Current_PDL_time);
    New_line;
  end if;
  -- *** DEMO MSGS ***

  -- Start the 10 second timer
  Emit (Start_timer, 10*sec);

  loop

    -- If no msgs are waiting, then
    if (Port_length (Recv_sensor_msg) = 0) AND
      (Port_length (Timer_alarm) = 0) then

      -- Wait 10 sec to send next status msg or
      -- until msg arrives from sensors
      Wait_for_activity;

    else

      -- If any msgs arrived from the sensors, then
      if Port_length (Recv_sensor_msg) > 0 then

        -- Check to see if any sensors report targets
        Targets_detected:= FALSE;
        for i in 1..Port_length (Recv_sensor_msg) loop
          Sensor_info:= Port_data (Recv_sensor_msg);

          -- *** DEMO MSGS ***
          if Current_debug_level > 20 then
            Put (CP_id);
            Put (" recvd msg from sensor");
            Put (Sensor_info.Initiator);
            Put ("; ");
            Put (Sensor_info.Number_of_targets);
            Put (" targets detected.");
            Put (" T=");
            Put (Current_PDL_time);
            New_line;
          end if;
          -- *** DEMO MSGS ***

          Consume (Recv_sensor_msg);
          if Sensor_info.Number_of_targets > 0 then
            Targets_detected:= TRUE;
```

```
        end if;
      end loop;

      -- If any sensors report targets detected, then
      if Targets_detected then

        -- If status had been peace, then change to war
        if Status.Order = DEFCON7 then
          Status.Order:= DEFCON1;

          -- Send status msg to all satellites
          Emit (Send_status_msg, Status);

          -- *** DEMO MSGS ***
          if Current_debug_level > 20 then
            Put (CP_id);
            Put (" broadcast status ");
            Put ("war.");
            Put (" T=");
            Put (Current_PDL_time);
            New_line;
          end if;
          -- *** DEMO MSGS ***

          -- Record the last time targets were seen
          Last_target_time:= Current_PDL_time;
        end if;
      end if;
    else
      -- Timer must have gone off

      -- Remove the Timer msg from the queue
      Consume (Timer_alarm);

      -- Send status msg to all satellites
      Emit (Send_status_msg, Status);

      -- *** DEMO MSGS ***
      if Current_debug_level > 20 then
        Put (CP_id);
        Put (" broadcast status ");
        if Status.Order = DEFCON1 then
          Put ("war.");
        elsif Status.Order = DEFCON7 then
          Put ("peace.");
        end if;
        Put (" T=");
        Put (Current_PDL_time);
        New_line;
      end if;
      -- *** DEMO MSGS ***
    end if;

    -- If at war and no targets are detected for 1 hr, then
    -- return to peaceful status.
    if Status.Order = DEFCON1 then
      if Current_PDL_time > Last_target_time + 60*min then
        Status.Order:= DEFCON7;
      end if;
    end if;
  end if;
end loop;
exception
  when others => Put_line("**Some error in Processor_init**");
$end;
```

```
--------------------------------------------------------------
--                    Timer Process
--------------------------------------------------------------

-- PROCESS SPECIFICATION

$process Timer is

  $parameter CP_id: string(1..7):= ("--CP--");

  $inport Start_timer: Time;

  $outport Sound_alarm: Boolean;

$end;

-- PORT LINKAGES
-- There are no port linkages within this process
-- Older versions of Sagen required the $link, this one doesn't

-- TASK SEMANTICS

$task Timer is

  Interval: Time;
  Wake_up: constant Boolean:= TRUE;

$begin

  wait_for_activity;
  Interval:= Port_data (Start_timer);
  consume (Start_timer);

  loop

    wait (Interval);
    emit (Sound_alarm, Wake_up);

  end loop;
$end;
```

-------- WEAPON PLATFORM --------

```
--------------------------------------------------
--          Weapons_Platform Platform Description
--------------------------------------------------

-- DESCRIPTION
--
--  The Weapons Platform (WP) listens to each incoming message.
--  If it is a command message, the WP gets the
--  current status from it. Only if it is at war does the
--  WP process messages from the sensor satellites. Furthermore, the WP
-- is concerned only with non-relayed sensor messages.
--
--  For each message processed, the WP finds the target with the highest
--  probability of kill (Pk). If that Pk is higher than some minimum, it
--  shoots that target.
--
```

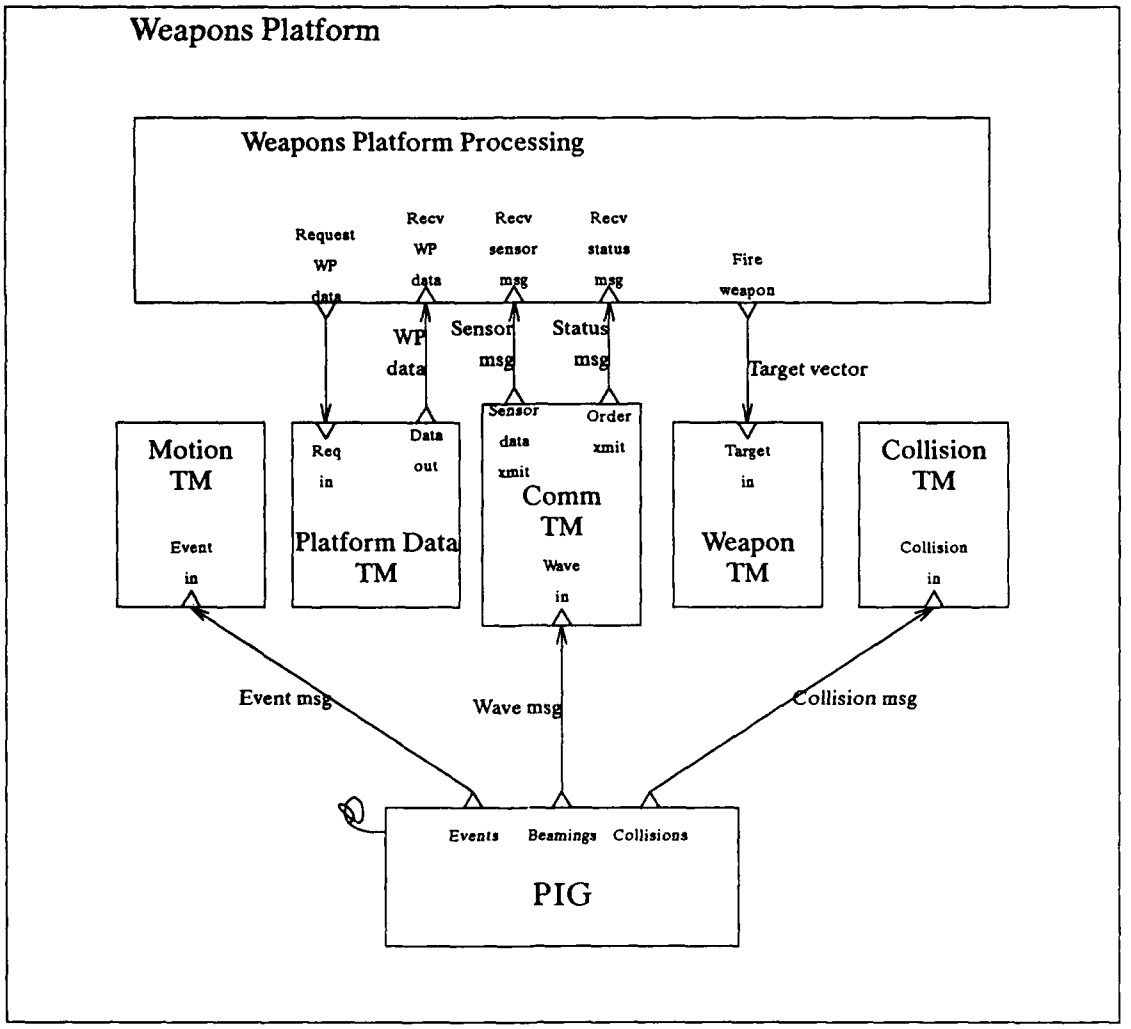


Figure 5-2. Weapons Platform Architecture.

```
--  The WP does not bother to count its remaining KEWs; it keeps playing
--  the game even after it has exhausted its supply.

-- PLATFORM SPECIFICATION
$platform Weapons_Platform:= Weapons_Platform is

  $parameter WP_id: string(1..7):= ("--WP--");

  $subprocess Weapons_Platform_Processing:= (WP_id),
        KKV_Weapon_TM:= (WP_id),
        Weapons_Platform_Communication_TM:= (WP_id),
        Platform_Data_TM:= (WP_id),
        Orbit_Equation_Of_Motion_TM:= (WP_id);
        -- The PIG is a predefined subprocess of every platform

  $subdata  Vector, Order, Track_data, Platform_data_req, Platform_data,
        Cone_msg, Event_Msg;

$end;

-- PORT LINKAGES

with Sensor_Cone_Response_TM_pkg;
$links Weapons_Platform is
$begin

  -- Exclude the Radar Return TM
  exclude_dyn_module(MYSELF,
    Sensor_Cone_Response_TM_pkg.Sensor_Cone_Response_TM_designator);

  -- Connect Weapons_Platform_Processing to KKV_Weapon_TM
  internal_link (Weapons_Platform_Processing.Fire_weapon,
          KKV_Weapon_TM.Target_in);

  -- Connect Weapons_Platform_Communication_TM to
  --   Weapons_Platform_Processing
  internal_link (Weapons_Platform_Communication_TM.Sensor_data_rcv,
          Weapons_Platform_Processing.Recv_sensor_msg);

  internal_link (Weapons_Platform_Communication_TM.Order_rcv,
          Weapons_Platform_Processing.Recv_status_msg);

  -- Connect Weapons_Platform_Processing to Platform_Data_TM
  internal_link (Weapons_Platform_Processing.Request_WP_data,
          Platform_Data_TM.Req_in);

  internal_link (Platform_Data_TM.Data_out,
          Weapons_Platform_Processing.Recv_WP_data);

  -- Connect PIG to Weapons_Platform_Communication_TM
  internal_link (PIG.Beamings,
          Weapons_Platform_Communication_TM.Cone_in);


  -- Connect PIG to Orbit_Equation_Of_Motion_TM
  internal_link (PIG.Events,
          Orbit_Equation_Of_Motion_TM.Event_in);

$end;


-- TASK SEMANTICS
-- The semantics of this platform are defined within its subprocesses
-- Older versions of Sagen required the $task, this one doesn't
```

```
--          Weapons_Platform_Processing Process
-------------------------------------------------------

-- THIS PROCESS USES
--
--   Procedure, Find_Pk, to determine probability of kill.
--   Function, Aim, to determine the vector for firing the weapon.
--
-- SIMULATION TIME USED
--
--   The computing time for this algorithm is (3 + 0.2/target) seconds.

-- PROCESS SPECIFICATION

$technology_module Weapons_Platform_Processing is

  $parameter WP_id: string(1..7):= ("--WP---");

  $inport  Recv_sensor_msg: Track_data,
       Recv_status_msg: Order,
       Recv_WP_data: Platform_data;

  $outport Fire_weapon: Vector,
       Request_WP_data: Platform_data_req;

$end;

-- PORT LINKAGES
-- There are no port linkages within this process
-- Older versions of Sagen required the $link, this one doesn't

-- TASK SEMANTICS

with Track_data_rec_ptr_pkg,Vector_IO;
use Track_data_rec_ptr_pkg,Vector_IO;
$task Weapons_Platform_Processing is

  Status_msg: Order;
  Status: Defense_status;
  Sensor_info: Track_data;
  Pk: Float:= 0.0;
  Pk_max: Float:= 0.0;
  WP_data: Platform_data;
  My_position: Vector;
  Best_target: Track_data_rec_ptr;
  Current_target: Track_data_rec_ptr;
  Target_vector: Vector;
  Seed: Integer:= 15;

  Pk_min: constant Float:= 0.2;
  sec: constant:= PDL_ticks_per_second;
  km: constant:= PDL_lunits_per_kilometer;

  procedure Find_Pk (
            Target: in Track_data_rec_ptr;
            Pk: out Float) is

    procedure Rand (Seed: in out integer; Num: out float) is
      -- Returns a float proportional to the random integer
      -- in the range 0.0 .. 1.0.

      A: constant:= 13849;
      M: constant:= 65536;
      C: constant:= 56963;

    begin
```

```
    Seed:= (seed * A + C) mod M;
    Num:= float(seed) * 1.52587890625E-5;
  end rand;

begin
  if Length (My_position - Target.Position) > Float(4500*km) then
    Pk:= 0.0;
  else
    Rand (Seed, Pk);
  end if;
end Find_Pk;


function Aim (
        Target: in Track_Data_rec_ptr;
        My_position: in Vector)
    return Vector is

  intercept_point: vector;
  new_target_position: vector;
  distance: float;
  delta_t: float;
  delta_d: float:= 1.0;

begin

  new_target_position:= target.position;
  while delta_d > 0.00001 loop
    intercept_point:= new_target_position;
    distance:= length(intercept_point - my_position);
    delta_t:= distance / (5.0 * Float(km/sec));
    new_target_position:= target.position + (delta_t * target.velocity);
    delta_d:= Length (new_target_position - intercept_point);
  end loop;
  return intercept_point;

end Aim;

$begin

  -- Start at peace
  Status:= DEFCON7;

  loop

    -- If no msgs are at any ports, then wait for msgs to come in
    if Port_length (Recv_status_msg) = 0 AND
      Port_length (Recv_sensor_msg) = 0 then
      Wait_for_activity;

    else
      -- If a status msg has arrived, then update the status
      if Port_length (Recv_status_msg) > 0 then
        Status_msg:= Port_data (Recv_status_msg);
        Consume (Recv_status_msg);
        Status:= Status_msg.Order;

        -- *** DEMO MSGS ***
        if Current_debug_level > 20 then
          Put (WP_id);
          Put (" recvd status ");
          if Status = DEFCON1 then
            Put ("war");
          elsif Status = DEFCON7 then
            Put ("peace");
          end if;
```

```
    Put (" from ");
    Put (Status_msg.Initiator);
    Put (". T=");
    Put (Current_PDL_time);
    New_line;
  end if;
  --- *** DEMO MSGS ***
end if;

--- If a sensor msg has arrived, then
if Port_length (Recv_sensor_msg) > 0 then
  Sensor_info:= Port_data (Recv_sensor_msg);
  Consume (Recv_sensor_msg);


  --- If the msg came directly from a sensor (i.e., not a relay) AND
  --- current status is DEFCON1, then
  if (Sensor_info.Track_list /= null) AND (Status = DEFCON1) then

    --- *** DEMO ***
    --- Get rid of the extra sensor msgs
    for i in 1..Port_length(Recv_sensor_msg) loop
      Consume (Recv_sensor_msg);
    end loop;
    --- *** DEMO ***

    --- If any targets were detected, then
    if Sensor_info.Number_of_targets > 0 then

      --- Get the current position of the Weapons Platform
      Emit (Request_WP_data, 0);
      Wait_for_activity ((1=>Recv_WP_data.Port));
      WP_data:= Port_data (Recv_WP_data);
      Consume (Recv_WP_data);
      My_position:= WP_data.Position;

      --- Pick the best target
      Pk_max:= 0.0;
      Current_target:= Sensor_info.Track_list;


      while (Current_target /= null) loop
        Find_Pk (Current_target, Pk);
        if Pk > Pk_max then
          Pk_max:= Pk;
          Best_target:= Current_target;
        end if;

        --- Get the next target
        Current_target:= Current_target.Next_rec;
      end loop;

      --- If the Pk is acceptable, then
      if Pk_max >= Pk_min then

        --- Aim at the selected target
        Target_vector:= Aim (Best_target, My_position);

        --- Shoot at the target
        Emit (Fire_weapon, Target_vector);

        --- *** DEMO MSGS ***
        if Current_debug_level > 20 then
          Put (WP_id);
          Put (" fired weapon toward target at ");
          Put (Target_vector);
```

```
            Put (". T=");
            Put (Current_PDL_time);
            New_line;
          end if;
          -- *** DEMO MSGS ***
        end if;
      end if;
    end if;
  end if;

  Wait (sec/2);
 end if;
end loop;
exception
  when others => write_process_full(MYSELF,"**Some error in ");
$end;
```

------- SENSOR PLATFORM --------

```
--------------------------------------------------
--        Sensor_Platform Platform Description
--------------------------------------------------

-- DESCRIPTION
--
--  The sensor platform scans the world and sends reports (to everyone)
--  about all detected targets. (Messages are sent regardless of whether
--  targets are detected or not. It also rebroadcasts summaries of
--  sensor messages originated by sensors of higher sensor-ID.

-- PLATFORM SPECIFICATION

$platform Sensor_Platform:= Sensor_Platform is

  $parameter SP_id: string(1..7):= ("--SP--");

  $subprocess Sensor_Platform_Processing:= (SP_id),
```

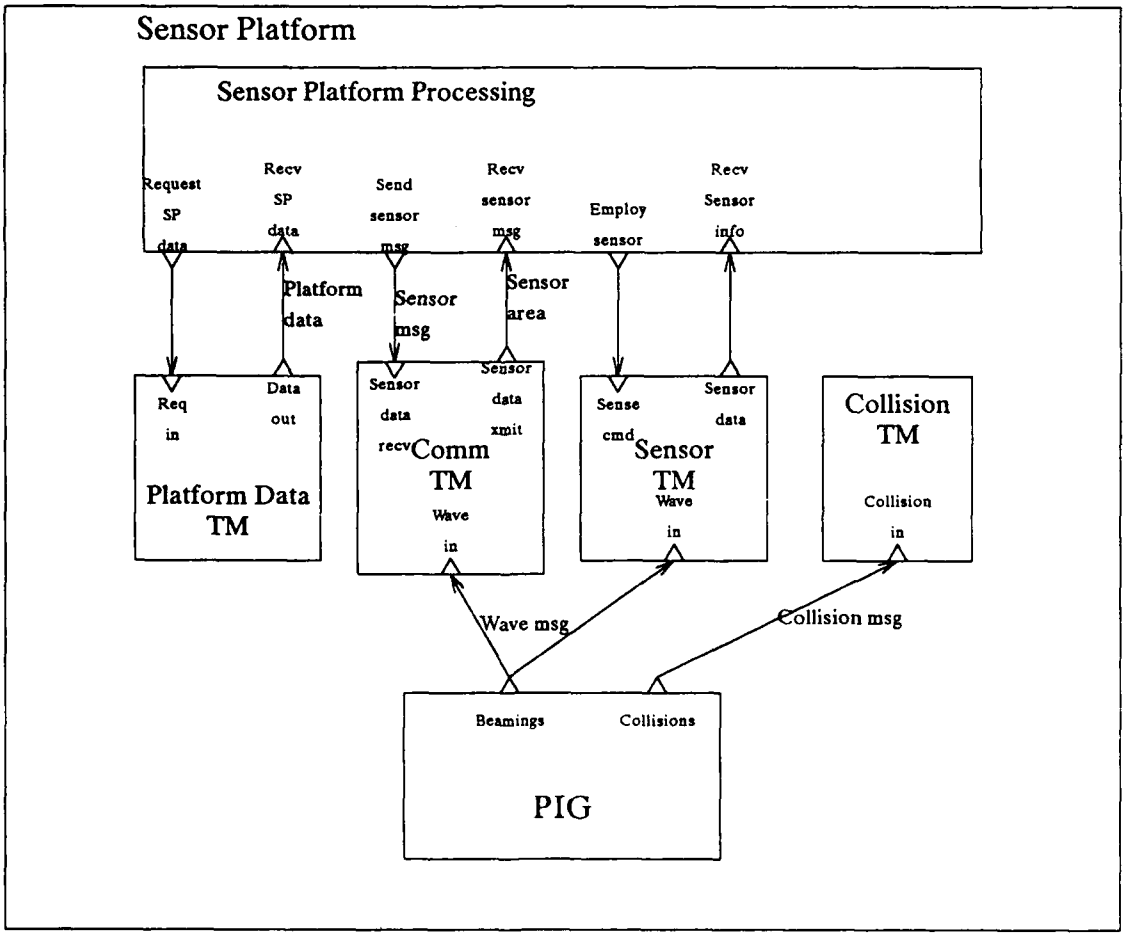


Figure 5-3. Sensor Platform Architecture.

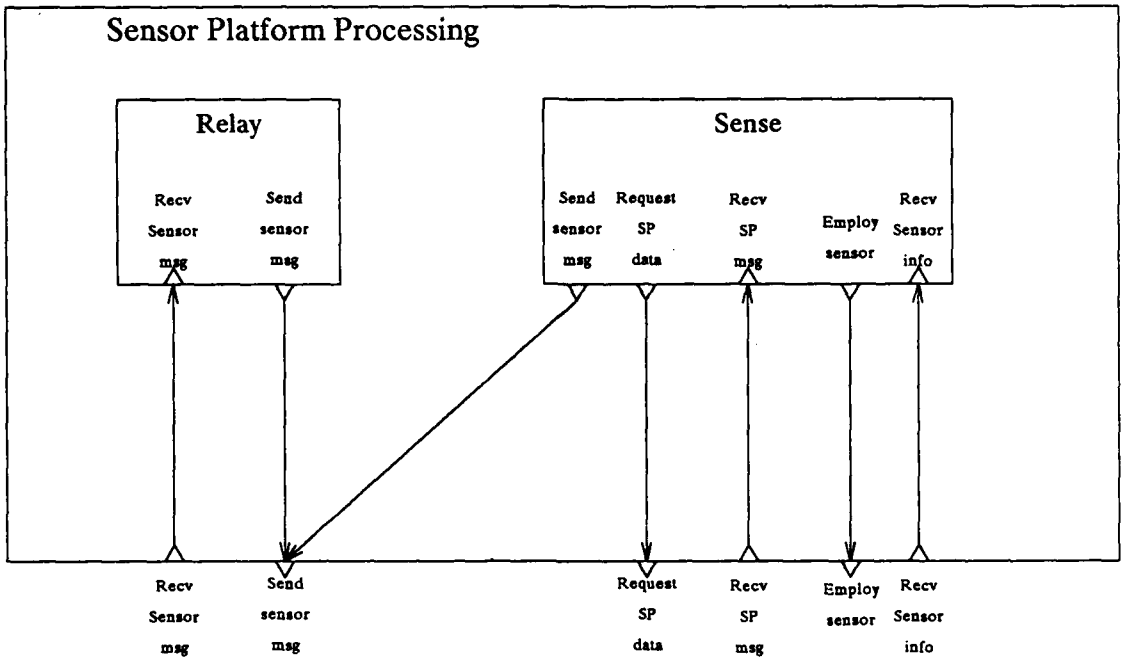


Figure 5-4. Next-Level Decomposition of Sensor Platform Processing

```
        Sensor_Device_TM:= (SP_id),
        Sensor_Satellite_Communication_TM:= (SP_id),
        Platform_data_TM:= (SP_id);
        -- The PIG is a predefined subprocess of every platform

  $subdata  Sense_req, Track_data, Platform_data_req, Platform_data,
        Cone_msg;

$end;


with Sensor_Cone_Response_TM_pkg;
$links Sensor_Platform is

$begin

  -- Exclude the Radar Return TM
  exclude_dyn_module(MYSELF,
    Sensor_Cone_Response_TM_pkg.Sensor_Cone_Response_TM_designator);

  -- Connect Sensor_Platform_Processing to Sensor_Device_TM
  internal_link (Sensor_Platform_Processing.Employ_sensor,
          Sensor_Device_TM.Sense_cmd);
  internal_link (Sensor_Device_TM.Sensor_data,
          Sensor_Platform_Processing.Recv_sensor_info);

  -- Connect Sensor_Platform_Processing to
  --   Sensor_Satellite_Communication_TM
  -- (Since the status msg is not used in this process, we
  -- don't need to link to the TM port that sends it.)
  internal_link (Sensor_Platform_Processing.Send_sensor_msg,
          Sensor_Satellite_Communication_TM.Sensor_data_xmit);
  internal_link (Sensor_Satellite_Communication_TM.Sensor_data_rcv,
          Sensor_Platform_Processing.Recv_sensor_msg);
```

```
— Connect Sensor_Platform_Processing to Platform_Data_TM
internal_link (Sensor_Platform_Processing.Request_SP_data,
        Platform_Data_TM.Req_in);
internal_link (Platform_Data_TM.Data_out,
        Sensor_Platform_Processing.Recv_SP_data);

— Connect PIG to Sensor_Satellite_Communication_TM
internal_link (PIG.Beamings,
        Sensor_Satellite_Communication_TM.Cone_in);

— Connect PIG to Sensor_Device_TM
internal_link (PIG.Beamings,
        Sensor_Device_TM.Cone_in);


$end;

— TASK SEMANTICS
— The semantics of this platform are defined within its subprocesses.
— Older versions of Sagen required the $task, this one doesn't

————————————————————————————————
—          Sensor_Platform_Processing Process
————————————————————————————————

— PROCESS SPECIFICATION

$process Sensor_Platform_Processing is

  $parameter SP_id: string(1..7):= ("—SP—");

  $inport   Recv_sensor_info: Track_data,
        Recv_sensor_msg: Track_data,
        Recv_SP_data: Platform_data;

  $outport  Employ_sensor: Sense_req,
        Send_sensor_msg: Track_data,
        Request_SP_data: Platform_data_req;

  $subprocess Sense:= (SP_id),
        Relay:= (SP_id);

$end;

— PORT LINKAGES

$links Sensor_Platform_Processing is

$begin

  — Connect Sense to parent (Sensor_Platform_Processing)
  inherited_link (Recv_sensor_info,
        Sense.Recv_sensor_info);
  inherited_link (Sense.Employ_sensor,
        Employ_sensor);
  inherited_link (Recv_SP_data,
        Sense.Recv_SP_data);
  inherited_link (Sense.Request_SP_data,
        Request_SP_data);
  inherited_link (Sense.Send_sensor_msg,
        Send_sensor_msg);

  — Connect Relay to parent (Sensor_Platform_Processing)
  inherited_link (Recv_sensor_msg,
        Relay.Recv_sensor_msg);
  inherited_link (Relay.Send_sensor_msg,
```

```
        Send_sensor_msg);
$end;

-- TASK SEMANTICS
-- The semantics are defined in the subprocesses Sense and Relay
-- Older versions of Sagen required the $task, this one doesn't

-------------------------------------------------------------
--                  Sense Process
-------------------------------------------------------------

-- TASK SPECIFICATION

$process Sense is

  $parameter SP_id: string(1..7):= ("--SP--");

  $inport   Recv_sensor_info: Track_data,
        Recv_SP_data: Platform_data;

  $outport  Employ_sensor: Sense_req,
        Request_SP_data: Platform_data_req,
        Send_sensor_msg: Track_data;
$end;

-- PORT LINKAGES
-- There are no port linkages within this process
-- Older versions of Sagen required the $link, this one doesn't

-- TASK SEMANTICS

with Vector_pkg;
use Vector_pkg;
$task Sense is

  SP_data: Platform_data;
  My_position: Vector;
  Sense_area: Sense_req;
  Sensor_info: Track_data;
  My_id: Integer:= 0;
  sec: constant:= PDL_ticks_per_second;

  function Get_int_id (Id_string: string) return Integer is
    begin
      case Id_string (7) is
        when '1' => return 1;
        when '2' => return 2;
        when '3' => return 3;
        when '4' => return 4;
        when others =>
          Put_line ("**** ERROR *** - Sense found incorrect id");
          return 0;
      end case;
    end Get_int_id;

$begin

  loop

    -- Get the current position of the Sensor Platform
    Emit (Request_SP_data, 0);
    Wait_for_activity ((1=>Recv_SP_data.Port));

    SP_data:= Port_data (Recv_SP_data);
    Consume (Recv_SP_data);
    My_position:= SP_data.Position;
```

```
   -- Set the current sense location
   -- Sense_area.Axis:= (0.0, 0.0, 0.0) - My_position
   --                   (0.0, 0.0, 0.0) - destination;
   -- Sense_area.Half_angle:= 360.0;
   -- Sense_area.Blackout_radiuse= 0.0;

   -- Send a msg to the sensor module requesting sensor data
   Emit (Employ_sensor, ( (0.0,0.0,0.0), (0.0,0.0,0.0), 360.0, 0.0) );

   -- Wait for the sensor data to be returned
   Wait_for_activity ((1=>Recv_sensor_info.Port));

   -- Read the data from the port
   Sensor_info:= Port_data (Recv_sensor_info);

   -- Remove that msg from the port queue
   Consume (Recv_sensor_info);
   end if;

   -- Complete the message by assigning the sensor id
   My_id:= Get_int_id (SP_id);
   Sensor_info.Initiator:= My_id;

   -- Send the msg
   Emit (Send_sensor_msg, Sensor_info);

   -- *** DEMO MSGS ***
   if Current_debug_level > 20 then
     Put (SP_id);
     Put (" sent sensor msg; ");
     Put (Sensor_info.Number_of_targets);
     Put (" targets detected.");
     Put (" T=");
     Put (Current_PDL_time);
     New_line;
   end if;
   -- *** DEMO MSGS ***

    Wait (2*sec);
  end loop;
  exception

 Put_line("**Some error in SPP_task**");
$end;

-------------------------------------------------------------
--                  Relay Process
-------------------------------------------------------------

-- PROCESS SPECIFICATION

$process Relay is

  $parameter SP_id: string(1..7):= ("--SP--");

  $inport    Recv_sensor_msg: Track_data;

  $outport   Send_sensor_msg: Track_data;

$end;

-- PORT LINKAGES
-- There are no port linkages within this process
-- Older versions of Sagen required the $link, this one doesn't

-- TASK SEMANTICS
```

```
$task Relay is

  Sensor_msg: Track_data;
  My_id: Integer:= 0;
  Initiator: Integer:=0;

  sec: constant:= PDL_ticks_per_second;

  function Get_int_id (Id_string: string) return Integer is
    begin
      case Id_string (7) is
        when '1' => return 1;
        when '2' => return 2;
        when '3' => return 3;
        when '4' => return 4;
        when others =>
          Put_line ("*** ERROR *** - Relay found incorrect id");
          return 0;
      end case;
    end Get_int_id;

$begin

  loop

    My_id:= Get_int_id (SP_id);

    -- If no sensor msgs have arrived, then wait for them.
    if Port_length (Recv_sensor_msg) = 0 then

      Wait_for_activity;

    else
      -- If a sensor msg has arrived from another platform, then
      if Port_length (Recv_sensor_msg) > 0 then

        -- Read the msg from the port
        Sensor_msg:= Port_data (Recv_sensor_msg);

        -- Remove that msg from the queue
        Consume (Recv_sensor_msg);

        -- If the id of the msg initiator is higher than my sensor id
        if Sensor_msg.Initiator > My_id then

          Sensor_msg.Initiator:= My_id;
          Sensor_msg.Track_list:= null;
          Emit (Send_sensor_msg, Sensor_msg);
        end if;
      end if;
    end if;

    Wait (1*sec);

  end loop;
  exception
$end;
```

6. Threat Architecture

The threat architecture is a *Russian_Missile_Base_Platform* that launches one missile every twenty seconds. The missile description is given in the *Russian_Missile_Platform* code. Basically, the missile is launched from the missile base by the *Russian_Missile_Launcher*. The launcher computes the target and the trajectory of the missile, and then creates a missile platform with the given trajectory.

The *Russian_Missile_Platform* contains two technology modules. One module returns sensor echos when beamed by a sensor, and the other module determines the result of collisions.

-------- RUSSIAN_MISSILE_BASE_PLATFORM --------

```
$platform Russian_Missile_Base_Platform:=Russian_Missile_Base is
  $subprocesses Russian_Missile_Launcher_TM:= (id), Russian_Missile_Base_BMC3:= (id);
  $parameter id:string(1..7):= ("R_BASE ");
  $subdata Target;
$end;


with Sensor_Cone_Response_TM_pkg;
$links Russian_Missile_Base_Platform is
$begin
  internal_link(Russian_Missile_Base_BMC3.target_out,
        Russian_Missile_Launcher_TM.target_in);
  exclude_dyn_module(MYSELF,
   Sensor_Cone_Response_TM_pkg.Sensor_Cone_Response_TM_designator);

  exception
    when others => put_line("***Some error in Russ_Miss_Base_Plat_init***");

$end;
```

-------- RUSSIAN_MISSILE_BASE_BMC3 --------

```
$process Russian_Missile_Base_BMC3 is
  $outport target_out:Target;
  $parameter platform_id:string(1..7):= ("------");
$end;

with math, random;
use math, random;
$task Russian_Missile_Base_BMC3 is
  target_lat: float;
  target_long: float;
  seed: integer:= 21;
  MISSILE_LIMIT: constant:= 20;
  SECONDS: constant:= PDL_ticks_per_second;
$begin
  for i in 1..MISSILE_LIMIT loop
    rand(seed,target_lat);
    target_lat:= target_lat * 15.0 + 30.0;
    rand(seed,target_long);
    target_long:= target_long * 45.0 + 240.0;
    emit(target_out,(target_lat,target_long));
    wait(PDL_duration_type(20 * SECONDS));
  end loop;

  exception
    when others => put_line("***Some error in Russ_Miss_Base_BMC3_TM_task***");
```

```
$end;
```

-------- RUSSIAN_MISSILE_LAUNCHER_TM --------

```
$technology_module Russian_Missile_Launcher_TM is
  $inport target_in:Target;
  $parameter platform_id:string(1..7):= ("------");
$end;

with Vector_pkg,
  Vector_IO,
  Math,
  Russian_Missile_Platform_pkg,
  latitude_n_longitude;
use Vector_pkg,Vector_IO;
$task Russian_Missile_Launcher_TM is
  use Russian_Missile_Platform_pkg,
    Russian_Missile_Platform_CP_pkg,
    latitude_n_longitude,
    Cones_n_Platforms.eqn_motion_pkg;
  use Russian_Missile_Platform_PARAM_pkg;
  use Math;
  SECONDS: constant:= PDL_ticks_per_second;
  KM: constant:= PDL_units_per_kilometer;
  current_target: Target;
  my_latitude: constant:= 55.8/180.0 * PI;
  my_longitude: constant:= 37.9/180.0 * PI;
  mu: constant:= (3.98633E5 * (KM ** 3)) / (SECONDS * SECONDS);
  missile_eom: eqn_motion_type;
  next_eom_rec: eqn_motion_type;
  init_pos: Vector;
  temp: float;
  flight_time: float;
  delta_t: PDL_duration_type;
  target_site: Vector;
  normal: Vector;
  perigee: Vector;
  inclination: float;
  omega: float;
  phi: float;
  xref, yref: float;
  delta_1: float;
  v_min: float;
  gamma: float;
  axis: float;
  eccentricity: float;
  mean_motion: float;
  theta: float;
  E, sin_of_E: float;
  tau: float;
  A11, A12, A13: float;
  A21, A22, A23: float;
  C1, C2: float;
  mean_anom: float;
  Ecc_Anom: float;
  missile_num: integer:= 1;
  missile_param: Russian_Missile_Platform_parameterization_ptr;
$begin
  init_pos:= location(my_latitude, my_longitude);
  temp:= length(init_pos);
  temp:= (temp + 1.01*KM)/temp;
  init_pos:= init_pos * temp;
  loop
```

```
wait_for_activity((1=>target_in.Port),StartTime => 0);
  current_target:= port_data(target_in);
  consume(target_in);
  target_site:= location(current_target.latitude,
                current_target.longitude);
  if current_debug_level > 10 then
    put("("); put(platform_id); put(")");
    put("(RMBLAUN) launch request: ");
    put(target_site);
    put_line(".");
  end if;
  normal.x:= init_pos.y * target_site.z
        - init_pos.z * target_site.y;
  normal.y:= init_pos.z * target_site.x
        - init_pos.x * target_site.z;
  normal.z:= init_pos.x * target_site.y
        - init_pos.y * target_site.x;
  inclination:= arccos(normal.z/length(normal));
  omega:= arctan( - normal.x / normal.y);
  phi:= arccos((init_pos * target_site)
       /(length(init_pos) * length(target_site))
      )/2.0;
  perigee:= (-1.0) * (init_pos + target_site);
  xref:= abs(normal.y);
  yref:= abs(normal.x);
  delta_1:= arccos(
          (xref * perigee.x + yref * perigee.y)
         /(sqrt(xref*xref + yref*yref) * length(perigee))
        );
  if perigee.z < 0.0 then
    delta_1:= delta_1 + PI;
  end if;
  v_min:= sqrt((2.0 * sin(phi)) / (1.0 + sin(phi)));
  gamma:= arcsin(cos(phi))/2.0;
  axis:= Re / (2.0 - v_min*v_min);
  eccentricity:= sqrt((1.0 - v_min*v_min*(2.0-v_min**2.0))
                * (sin(gamma)**2.0));
  mean_motion:= sqrt(mu/(axis**3.0));
  theta:= pi - phi;
  sin_of_E:= sqrt(1.0 - eccentricity*eccentricity)*sin(theta)
         /(1.0 + eccentricity * cos(theta));
  E:= arcsin(sin_of_E);
  tau:= (E - eccentricity*sin_of_E)/mean_motion;
  flight_time:= 2.0 * PI / mean_motion - 2.0*tau;
  A11:= cos(delta_1) * cos(omega)
     - sin(delta_1) * cos(inclination) * sin(omega);
  A12:= cos(delta_1) * sin(omega)
     + sin(delta_1) * cos(inclination) * cos(omega);
  A13:= sin(delta_1) * sin(inclination);
  A21:= - sin(delta_1) * cos(omega)
      - cos(delta_1) * cos(inclination) * sin(omega);
  A22:= cos(delta_1) * cos(inclination)* cos(omega)
     - sin(delta_1) * sin(omega);
  A23:= cos(delta_1) * sin(inclination);
  delta_t:= PDL_duration_type(flight_time / 20.0);
  missile_eom:= new_eqn_motion_rec;
  missile_eom.delta_t:= delta_t;
  next_eom_rec:= missile_eom;
  for j in 1..19 loop
    mean_anom:= mean_motion*(float(j*integer(delta_t)) + tau);
    Ecc_Anom:= mean_anom
      + eccentricity * sin(mean_anom)
      + eccentricity * eccentricity * sin(2.0*mean_anom)/2.0
      + (eccentricity ** 3.0)/2.0
          * sin(mean_anom) ** 2.0 * cos(mean_anom)
      + (eccentricity ** 4.0)/6.0
```

```
            * sin(mean_anom) ** 3.0 * cos(mean_anom);
        c1:= axis * (cos(Ecc_Anom) - eccentricity);
        c2:= axis * sqrt(1.0 - eccentricity * eccentricity)
            * sin(Ecc_Anom);
        next_eom_rec.position.x:= a11*c1 + a21*c2;
        next_eom_rec.position.y:= a12*c1 + a22*c2;
        next_eom_rec.position.z:= a13*c1 + a23*c2;
        if j < 19 then
          next_eom_rec.next_rec:= new_eqn_motion_rec;
          next_eom_rec:= next_eom_rec.next_rec;
          next_eom_rec.delta_t:= delta_t;
        end if;
      end loop;
      missile_param:= new Russian_Missile_Platform_parameterization;
      missile_param.id:= "RMiss ";
      put(missile_param.id(6..7),missile_num);
      if current_debug_level > 50 then
        put("("); put(platform_id); put(")");
        put_line("(RMBLAUN) Launching missile now.");
      end if;
      Russian_Missile_Platform_CP_pkg.create_platform(
              Russian_Missile_Platform_designator,
              param          => missile_param,
              initial_position => init_pos,
              eqn_motion         => missile_eom,
              expected_lifetime=>
                PDL_duration_type(flight_time));
      missile_num:= missile_num + 1;
      wait(0);
  end loop;

  exception
    when others => put_line("***Some error in Russ_Miss_Launch_TM_task***");

$end;
```

-------- RUSSIAN_MISSILE_PLATFORM --------

```
$platform Russian_Missile_Platform:=Russian_Missile is
  $subprocesses Russian_Missile_TM:= (id);
  $parameter id:string(1..7):= ("missile");
  $subdata Platform_Msg,Event_Msg;
$end;

with Platform_Collision_TM_pkg;
$links Russian_Missile_Platform is
$begin
  internal_link(PIG.collisions,Russian_Missile_TM.part_in);
  internal_link(PIG.events   ,Russian_Missile_TM.event_in);
  exclude_dyn_module(MYSELF,
    Platform_Collision_TM_pkg.Platform_Collision_TM_designator);


  exception
    when others => put_line("***Some error in Russ_Miss_Plat_init***");

$end;
```

-------- RUSSIAN_MISSILE_TM --------

```
$technology_module Russian_Missile_TM is
  $import part_in:Platform_Msg,
      event_in:Event_Msg;
  $parameter platform_id:string(1..7):= ("------");
$end;

with Vector_pkg,Vector_IO;
use Vector_pkg,Vector_IO;
$task Russian_Missile_TM is
  use Cones_n_Platforms.interface_procs;
  SECONDS: constant:= PDL_ticks_per_second;
  pos: vector;
  which_port: integer;
$begin
  loop
    wait_for_activity((event_in.Port,part_in.Port),which_port,
              StartTime => 0,Time_out=>SECONDS);
    if which_port=1 then
       consume(event_in);
       put("("); put(platform_id); put(")");
       put_line("(RMISSTM) Now I'm dead (R.I.P.)");
       destroy_self;
    end if;
    if which_port=2 then
      consume(part_in);
      destroy_self;
    end if;
    if current_debug_level > 20 then
      put("("); put(platform_id); put(")");
      put("(RMISSTM) position update: ");
      pos:= platform_position;
      put(pos);
      put(" at t = ");
      put(Current_PDL_time,1); put(".");
      new_line;
    end if;
  end loop;

  exception
    when others => write_process_full(MYSELF,"***Some error in ","***");
$end;
```

7. Technology Modules

The following contains all the technology modules referenced in the preceding SAGEN code. This includes the *KKV_Platform*, the *Communication* modules, the *Sensor_Response* module, and others.

7.1. KKV Technology

The KKVs are launched by the *KKV_Weapon_TM* technology module. This module will launch up to 12 KKVs. The *KKV_Platform* represents a projectile. This platform contains two technology module. The *Platform_Collision_TM* determines the result of a collision and the *Tracker_TM* periodically prints the location of the KKV.

-------- KKV_WEAPON_TM --------

```
with KKV_Platform_pkg;
$technology_module KKV_Weapon_TM is
  $inport target_in:Vector;
  $parameter platform_id:string(1..7):= ("--KKV--");
$end;


with Vector_IO;
use Vector_IO;
$task KKV_Weapon_TM is
  use Cones_n_Platforms.interface_procs,Cones_n_Platforms.eqn_motion_pkg,
    KKV_Platform_pkg,KKV_Platform_pkg.KKV_Platform_CP_pkg;
  use KKV_Platform_PARAM_pkg;
  KM: constant:= PDL_Junits_per_kilometer;
  SECONDS: constant:= PDL_ticks_per_second;
  current_target: Vector;
  KKV_LIMIT: integer:= 12;
  kkv_param: KKV_Platform_parameterization_ptr;
  kkv_id: string(1..7):= "KKV-#--";
  kkv_eom: eqn_motion_type;
  init_pos: point_type;
  distance: float;
  flight_time: PDL_duration_type;
  temp: float;

$begin
  while KKV_LIMIT > 0 loop
    wait_for_activity((1=>target_in.Port),StartTime => 0);
      current_target:= port_data(target_in);
      consume(target_in);
      init_pos:= platform_position;
      temp:= length(init_pos);
      temp:= (temp - 1.01*KM)/temp;
      init_pos:= init_pos * temp;
      distance:= length(current_target - init_pos);
      flight_time:= PDL_duration_type(integer(
              distance / float(5 * KM / SECONDS)
             ));
      kkv_eom:= new_eqn_motion_rec;
      kkv_eom.position:= current_target;
      kkv_eom.delta_t:= flight_time;
      kkv_eom.next_rec:= null;
      kkv_param:= new KKV_Platform_parameterization;
      kkv_param.owner:= platform_id;
      put(kkv_id(6..7),13 - KKV_LIMIT);
```

```
      kkv_param.id:= kkv_id;
      KKV_Platform_CP_pkg.create_platform(KKV_Platform_designator,
              param          => kkv_param,
              initial_position => init_pos,
              eqn_motion       => kkv_eom,
              expected_lifetime => flight_time);
      KKV_LIMIT:= KKV_LIMIT - 1;
    wait(0);
  end loop;
  loop
    wait_for_activity((1=>target_in.Port),StartTime => 0);
      consume(target_in);
    wait(0);
  end loop;

  exception
    when others => put_line("***Some error in KKV_Weapon_TM_task***");

$end;
```

-------- KKV_PLATFORM_PKG --------

```
$platform KKV_Platform:=KKV is
  $parameters owner:string(1..7):= ("------"), id:string(1..7):= ("-KKV-");
$end;

with Sensor_Cone_Response_TM_pkg;
$links KKV_Platform is
$begin
  exclude_dyn_module(MYSELF,
    Sensor_Cone_Response_TM_pkg.Sensor_Cone_Response_TM_designator);


  exception
    when others => put_line("***Some error in KKV_Plat_init***");
$end;
```

-------- TRACKER_TM --------

```
$dynamic_tech_module Tracker_TM:= TRACKER_TM is
$end;

with Vector_pkg,Vector_IO;
use Vector_pkg,Vector_IO;
$task Tracker_TM is
  use Cones_n_Platforms.interface_procs;
  use FLT_IO;
  pos: vector;
  Seconds: constant:= PDL_ticks_per_second;
$begin
  loop
    wait(Seconds);
    if current_debug_level >= 0 then
      put("("); write_process_full(MYSELF,"(",")",end_of_line=>false);
      put(" current position: ");
      pos:= platform_position;
      put(pos);
      put(" at t = ");
```

```
      put(float(Current_PDL_time)/float(Seconds),3,4,0);
      put_line(" seconds.");
    end if;
    wait(0);
  end loop;

  exception
    when others => write_process_full(MYSELF,"***Some error in ","***");
$end;
```

7.2. Communications Technology

The communication technology is used to send and receive messages via SADMT cones. The *Ground_Station_Communication_TM* transmits orders from the *Command_Post* and receives sensor data from the *Sensor_Platforms*.

-------- GROUND_STATION_COMMUNICATION_TM --------

```
$technology_module Ground_Station_Communication_TM is
  $inport cone_in:Cone_Msg,
      order_xmit:Order;
  $outport sensor_data_rcv:Track_Data ;
  $parameter platform_id:string(1..7):= ("------");
  $cone Order_ptr;
$end;

with Track_Data_ptr_pkg;
$task Ground_Station_Communication_TM is
  use Track_Data_ptr_pkg;
  current_order: Order;
  o_ptr: Order_ptr;
  wmsg: Cone_Msg;
  sensor_data. Track_Data;
  d_ptr: Track_Data_ptr;
  m_ptr: PDL_magic_ptr;
$begin
  o_ptr:= new Order;
  loop
    wait_for_activity((order_xmit.Port,cone_in.Port),StartTime => 0);
    if Port_length(order_xmit) /= 0 then
      current_order:= port_data(order_xmit);
      consume(order_xmit);
      o_ptr.all:= current_order;
      wait(1);
      create_cone(Command_Transmission, data => o_ptr);
    elsif Port_length(cone_in) /= 0 then
      wmsg:= port_data(cone_in);
      consume(cone_in);
      if wmsg.designator = Sensor_Data_Transmission then
        d_ptr:= cast_magic_ptr_into_track_data_ptr(wmsg.data);
        sensor_data:= d_ptr.all;
        wait(1);
        emit(sensor_data_rcv, sensor_data);
      end if;
    end if;
    wait(0);
  end loop;

  exception
    when others => put_line("***Some error in Gnd_Stat_Com_TM_task***");

$end;
```

The *Sensor_Satellite_Communication_TM* broadcasts sensor data to all other platforms, and it receives orders from the *Command_Post* and sensor data from other sensors.

-------- SENSOR_SATELLITE_COMMUNICATION_TM --------

```
$technology_module Sensor_Satellite_Communication_TM is
  $inport cone_in:Cone_Msg,
      sensor_data_xmit:Track_Data;
  $outport order_rcv:Order,
      sensor_data_rcv:Track_Data ;
  $parameter platform_id:string(1..7):= ("------");
  $cone Track_Data_ptr;
$end;

with Order_ptr_pkg, Track_Data_ptr_pkg;
$task Sensor_Satellite_Communication_TM is
  use Order_ptr_pkg, Track_Data_ptr_pkg;
  wmsg: Cone_Msg;
  current_order: Order;
  o_ptr: Order_ptr;
  sensor_data: Track_Data;
  d_ptr: Track_Data_ptr;
  which_port: integer;
$begin
  d_ptr:= new Track_Data;
  loop
    wait_for_activity((sensor_data_xmit.Port,cone_in.Port),which_port,StartTime => 0);
    if which_port=1 then
      sensor_data:= port_data(sensor_data_xmit);
      consume(sensor_data_xmit);
      d_ptr.all:= sensor_data;
      wait(1);
      create_cone(Sensor_Data_Transmission, data => d_ptr);
    elsif which_port = 2 then
      wmsg:= port_data(cone_in);
      consume(cone_in);
      if wmsg.designator = Command_Transmission then
        o_ptr:= cast_magic_ptr_into_order_ptr(wmsg.data);
        current_order:= o_ptr.all;
        wait(1);
        emit(order_rcv,current_order);
      elsif wmsg.designator = Sensor_Data_Transmission then
        d_ptr:= cast_magic_ptr_into_track_data_ptr(wmsg.data);
        sensor_data:= d_ptr.all;
        wait(1);
        emit(sensor_data_rcv, sensor_data);
      end if;
    end if;
    wait(0);
  end loop;

  exception
    when others => put_line("***Some error in Sensor_Sat_Com_TM_task***");

$end;
```

The *Weapons_Platform_Communication_TM* receives orders from the *Command_Post*s and target information form the *Sensor_Platform*s.

-------- PLATFORM_COMMUNICATION_TM --------

```
$technology_module Weapons_Platform_Communication_TM is
  $inport cone_in:Cone_Msg;
  $outport order_rcv:Order,
       sensor_data_rcv:Track_Data;
  $parameter platform_id:string(1..7):= ("------");
$end;

with Order_ptr_pkg, Track_Data_ptr_pkg;
$task Weapons_Platform_Communication_TM is
  use Order_ptr_pkg, Track_Data_ptr_pkg;
  wmsg: Cone_Msg;
  current_order: Order;
  o_ptr: Order_ptr;
  sensor_data: Track_Data;
  d_ptr: Track_Data_ptr;
$begin
  loop
    wait_for_activity((1=>cone_in.Port),StartTime => 0);
      wmsg:= port_data(cone_in);
      consume(cone_in);
      if wmsg.designator = Command_Transmission then
        o_ptr:= cast_magic_ptr_into_order_ptr(wmsg.data);
        current_order:= o_ptr.all;
        wait(1);
        emit(order_rcv, current_order);
      elsif wmsg.designator = Sensor_Data_Transmission then
        d_ptr:= cast_magic_ptr_into_track_data_ptr(wmsg.data);
        sensor_data:= d_ptr.all;
        wait(1);
        emit(sensor_data_rcv, sensor_data);
      end if;
    wait(0);
  end loop;

  exception
    when others => put_line("****Some error in Platform_Com_TM_task****");

$end;
```

7.3. Other Technology Modules

The *Orbit_Equation_of_Motion_TM* is a stub process. Currently this process acknowledge the SADMT *Event_Msg*; however, a new equation of motion is not generated by this TM.

-------- ORBIT_EQUATION_OF_MOTION_TM --------

```
$technology_module Orbit_Equation_of_Motion_TM is
  $inport event_in:Event_Msg;
  $parameter platform_id:string(1..7):= ("------");
$end;


$task Orbit_Equation_of_Motion_TM is
  emsg: Event_Msg;
$begin
  loop
    wait_for_activity((1=>event_in.Port),StartTime => 0);
      emsg:= port_data(event_in);
      consume(event_in);
      if emsg = end_of_eqn_motion then
        put_line("(end-of-eqn-motion).");
        put_line("            " &
          " Computing new eqn-of-motion now.");
      else
        put_line("(NOT end-of-eqn-motion).");
      end if;
    wait(0);
  end loop;

  exception
    when others => put_line("***Some error in Orbit_Eqn_Mot_TM_task***");

$end;
```

The *Platform_DATA_TM* returns information concerning physical properties of the platform. This includes the platforms mass, equation of motion, speed, and so on.

-------- PLATFORM_DATA_TM --------

```
$technology_module Platform_Data_TM is
  $inport req_in:Platform_Data_Req;
  $outport data_out:Platform_Data;
  $parameter platform_id:string(1..7):= ("------");
$end;

with Vector_pkg,Vector_IO;
use Vector_pkg,Vector_IO;
$task Platform_Data_TM is
  use Cones_n_Platforms.interface_procs;
  use FLT_IO;
  part_data: Platform_Data;
  phys_block: physical_stuff_block;
$begin
  loop
    wait_for_activity((1=>req_in.Port),StartTime => 0);
      consume(req_in);
      phys_block:= get_physical_stuff;
      part_data.designator:= get_my_type;
```

```
      part_data.mass:= phys_block.mass;
      part_data.eqn_motion:= phys_block.eqn_motion;
      part_data.current_eqn_motion_segment:=
        phys_block.current_eqn_segment;
      part_data.when_arrived_this_segment:=
        phys_block.when_arrived_this_segment;
      part_data.when_leaving_this_segment:=
        phys_block.when_leaving_this_segment;
      part_data.position:= platform_position;
      part_data.speed:= phys_block.speed_this_segment;
      part_data.velocity:=
        (phys_block.where_at_end_of_segment -
         phys_block.where_at_start_of_segment) /
         phys_block.delta_t_this_segment;
      emit(data_out,part_data);
    wait(0);
  end loop;


  exception
    when others => put_line("***Some error in Platform_Info_TM_task***");

$end;
```

The *Sensor_Cone_Response_TM* listens to all beamings and ignores all but the *Sensor_Cone* beamings. When a *Sensor_Cone* beaming is detected, this TM returns a *Sensor_Cone_Reflection* to the platform that originated the *Sensor_Cone*.

-------- SENSOR_CONE_RESPONSE_TM --------

```
$dynamic_tech_module Sensor_Cone_Response_TM:= SENSOR_CONE_RESPONSE is
  $cone_inport cone_in;
  $cone Track_Data_rec_ptr;
$end;

with Vector_IO,Sensor_Cone_pkg;
with Track_Data_rec_ptr_pkg;
use Vector_IO,Sensor_Cone_pkg;
$task Sensor_Cone_Response_TM is
  use Cones_n_Platforms.interface_procs;
  wmsg: Cone_Msg;
  part_data: Track_Data_rec_ptr;
  eom: eqn_motion_type;
$begin
  part_data:= new Track_Data_rec;
  part_data.next_rec:= null;
  loop
    wait_for_activity((1=>cone_in.Port),StartTime => 0);
      wmsg:= port_data(cone_in);
      consume(cone_in);
      if wmsg.designator = Sensor_Cone then
        part_data.position:= platform_position;
        eom:= null;
        part_data.velocity:= (0.0,0.0,0.0);

        create_cone(Sensor_Cone_Reflection,
              RetAddr => wmsg.initiator_id,
              data => part_data);
      end if;
    wait(0);
  end loop;
  exception
```

```
    when others => write_process_full(MYSELF,"***Some error in ","***");
$end;
```

The *Sensor_Device_TM* waits for a signal to arrive on its *sense_cmd* port. When the signal arrives, it transmits a *Sensor_Cone* and wait for echos. As the echos are received, a track file is constructed. After all of the echos have been received, the track file is passed over the outport to the sensor platform.

-------- SENSOR_DEVICE_TM --------

```
$technology_module Sensor_Device_TM is
  $inport cone_in:Cone_Msg,
      sense_cmd:Sense_Req;
  $outport sensor_data:Track_Data;
  $parameter platform_id:string(1..7):= ("------");
  $cone Sense_Req_ptr;
$end;

with Vector_IO,Track_Data_rec_ptr_pkg,Sensor_Cone_pkg;
use Vector_IO,Track_Data_rec_ptr_pkg,Sensor_Cone_pkg;
$task Sensor_Device_TM is
  use Cones_n_Platforms.interface_procs;
  use FLT_IO;
  cmd: Sense_Req;
  wmsg: Cone_Msg;
  data: Track_Data_rec;
  track_info: Track_Data;
  new_node: Track_Data_rec_ptr:= null;
$begin
  loop
    if Port_length(sense_cmd) = 0 and then
     Port_length(cone_in  ) = 0 then
       wait_for_activity;
    elsif Port_length(cone_in) /= 0 then
      consume(cone_in);
    else
      cmd:= port_data(sense_cmd);
      consume(sense_cmd);
      create_cone(Sensor_Cone, cone=>cmd);
      wait(1);
      track_info.track_list:= null;
      track_info.number_of_targets:= 0;
      while Port_length(cone_in) /= 0 loop
        wmsg:= port_data(cone_in);
        consume(cone_in);
        if wmsg.designator = Sensor_Cone_Reflection then
          new_node:= cast_magic_ptr_into_track_data_rec_ptr(
            wmsg.data);
          data:= new_node.all;
          new_node:= new_track_data_rec;
          new_node.all:= data;
          new_node.next_rec:= track_info.track_list;
          track_info.track_list:= new_node;
          track_info.number_of_targets:=
            track_info.number_of_targets + 1;
        wait(1);
      end loop;
      emit(sensor_data,track_info);
      if track_info.number_of_targets = 0 then
        if current_debug_level > 60 then
          put("("); put(platform_id); put(")");
          put("(SENSORD) No targets; killing of platform now.");
```

```
        end if;
        destroy_self;
      end if;
    end if;
    wait(0);
  end loop;

  exception
    when others => put_line("***Some error in Sensor_Device_TM***");

$end;
```

The *Platform_Collision_TM* is simple. This TM destroys the host platform as soon as a collision is detected.

-------- PLATFORM_COLLISION_TM --------

```
$dynamic_tech_module Platform_Collision_TM:= PLATFORM_COLLISION_TM is
  $platform_inport part_in;
$end;

with Vector_pkg,Vector_IO;
use Vector_pkg,Vector_IO;
$task Platform_Collision_TM is
  use Cones_n_Platforms.interface_procs;
  pos: vector;
$begin
  loop
    wait_for_activity((1=>part_in.Port),StartTime => 0);
      consume(part_in);
      write_process_full(MYSELF,"(",")",false);
      put_line("(PARTCOL) Now I'm dead (R.I.P.)");
      destroy_self;
    wait(0);
  end loop;
  exception
    when others => write_process_full(MYSELF,"***Some error in ","***");
$end;
```

Distribution List for P-2036

Sponsor

LTC Jon Rindt — 1 copy
SDIO
Pentagon
BM/C3
1E149
Washington, DC 20301-7100

CAPT David Hart — 1 copy
SDIO
Pentagon
BM/C3
1E149
Washington, DC 20301-7100

LTC Chuck Lillie — 1 copy
SDIO
Pentagon
BM/C3
1E149
Washington, DC 20301-7100

LTC Pete Sowa — 1 copy
SDIO
Pentagon
BM/C3
1E149
Washington, DC 20301-7100

Other

Defense Technical Information Center — 2 copies
Cameron Station
Alexandria, VA 22314

(Each should receive 1 copy.)

F.R. Albrow
MoD (PE) RSRE
St. Andrews Road
Great Malvern
Worcester WR14 3PS
England

John Anderson
11569 Hicks Court
Manassas, VA 22111

Jim Armitage
GTE SSD
1 Research Dr.
Westborough, MA 01581

David Audley, Associate
Financial Strategies
Prudential Bache Securities
26th Floor
199 Water
New York, NY 10292

Dr. Algirdas Avizienis
Computer Science Department
4731 Boelter Hall
University of California, Los Angeles
Los Angeles, CA 90024

Dan Baker
TASC
55 Walkers Brook Drive
Redding, MA 01867

Gary H. Barber
Program Manager
Intermetrics, Inc.
1100 Hercules, Suite 300
Houston, TX 77058

John Barry
SJ-72
Rockwell
P.O. Box 3644
Seal Beach, CA 90740-7644

Elizabeth Bently
Marketing Coordinator
Research Triangle Institute
P.O. Box 12184
Research Triangle Park, N.C. 27709

Cheryl Bittner
General Electric
Box 8555, Bldg. 19, Suite 200
Philadelphia, PA 19101

Grady Booch
Director, Software Engineering Program
Rational
1501 Salado Dr.
Mountain View, CA 94043

Gina Bowden
MS 202
Teledyne Brown Engineering
Cummings Research Park
Huntsville, AL 35807

James M. Boyle
Mathematics & Computer Science Division
Argonne National Laboratory
Argonne, IL 60549-4844

Craig Bredin
GTE Strategic Systems Division
3322 South Memorial Parkway
Suite 53
P.O. Box 14009
Huntsville, AL 35185

Capt. John R. Brill
USAF Space Division/ALR
DET 3 AFALC
Box 92960 WPC
Los Angeles, CA 90009

Alton L. Brintzenhoff
Manager, Ada Technology Operating Center
Syscon Corporation
3990 Sherman St.
San Diego, CA 92110

Dr. James C. Browne
Computation Center
Department of Computer Science
University of Texas at Austin
Austin, TX 78712

Dr. Robert R. Brown
Rand Corp.
1700 Main St.
P.O. Box 2138
Santa Monida, CA 90406

Miguel Carrio
Teledyne Brown Engineering
3700 Pender Dr.
Fairfax, VA 22030

Tom Cashion
CALSPAN Corporation
P.O. Box 9X
Lexington, MA 02123

Virginia Castor, Director
Ada Joint Program Office
1211 Fern St., Room C-107
Arlington, VA 22202

Yvonne Cekel
Marketing Services Manager
Cadre Technologies, Inc.
222 Richmond St.
Providence, RI 02903

Bijoy G. Chatterjee
Deputy Director
Advanced Technology Systems
ESL/TRW
495 Java Dr.
P.O. Box 3510
Sunnyvale, CA 3510

John L. Chinn
Manager, Advanced Technology program
General Electric Space Systems Division
4041 North First St.
San Jose, CA 95134

Starla Christakos
AFATL/SAI
Eglin Air Force Base, FL 320542

Dr. Joe Clema
ECAC/IITRI
185 Admiral Cochrane Drive
Annapolis, MD 21401

Karen Coates
Program Manager, Advanced Technology
ESL/TRW
495 Java Dr.
P.O. Box 3510
Sunnyvale, CA 94088-3510

Susan Coatney
Information Science Institute
University of Southern California
4676 Admiralty Way
Marina del Ray, CA 90292-5950

Mr. Danny Cohen
Director, System Division
Information Science Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292-6695

Christopher F. Cole
Marketing Representative
Technology Programs
General Electric Company
Space Systems Division
Valley Forge Space Center
P.O. Box 8555
Philadelphia, PA 19101

Carol Combs
National Security Agency
9800 Savage Road
Ft. Meade, MD 20755-6000

Edward R. Comer
Software Productivity Solutions, Inc.
122 North 4th Av.
Indialantic, FL 32903

Lawrence L. Cone
Cone Software Laboratory
312 East Summit Av.
Haddonfield, N.J. 08033

Robert P. Cook
Department of Computer Science
Thornton Hall
University of Virginia
Charlottesville, VA 22903

Chuck Cooper
Control Data Corporation
901 E. 78th Street
MS BMW 03M
Minneapolis, MN 55420

Lee Cooper
Advanced Technology
2121 Crystal Drive, Suite 200
Arlington, VA 22202

Mark Cosby
Science Applications International Corp.
1710 Goodridge Drive
McLean, VA 22012

L. Cristina
USASD
ATTN: DASD-H-SBY
P.O. 1500
106 Wynn Dr.
Huntsville, AL 35807-3801

Vincent Dambrauskas
Technical Director
Washington Technical Center
Strategic Systems Division
GTE Government Systems Corporation
6850 Versar Center, Suite 354
Springfield, VA 22151-4196

Samuel A. DeNitto
Romse Air Development Center
RADC/COE, Bldg. 3
Griffis AFB, NY 13441-5700

Cameron M.G. Donaldson
Software Productivity Solutions, Inc.
122 North 4th Av.
Indialantic, FL 32903

Ralph Duncan
Control Data Government Systems
300 Embassy Row
Atlanta, GA 30328

Stephen Edwards
1-55 Caltech
Pasadena, CA 91126

Jim Egolf
Ford Aerospace & Computer Corp.
10440 State Hwy. 83
Colorado Springs, CO 80908

David A. Fisher
Incremental Systems Corporation
319 S. Craig St.
Pittsburgh, PA 15213

Dave Fittz
STARS Program Office
OUS
OUSDRE (R&AT/CET)
3D139
1211 Fern St., C-112
Washington, D.C. 20301-3081

Michel A. Floyd
Integrated Systems Inc.
101 University Av.
Palo Alto, CA 94301-1695

Richard Frase
SRS Technologies
1500 Wilson Blvd., Suite 800
P.O. Box 12707
Arlington, VA 22209-8707

George Gearn
Applied Researsch & Engineering
7 Railroad Avenue, Suite F
Bedford, MA 01730

Victor Giddings
MITRE Corporation
Burlington Road
Bedford, MA 01730

Claren Giese
SDIO
Pentagon
T/KE
Washington, DC 20301-7100

Colin Gilyeat
Advanced Technology
2121 Crystal Drive
Arlington, VA 22202

Robert T. Goettge
Advanced Systems Technology
12200 East Briarwood Av.
Suite 260
Englewood, CO 80112

H.T. Goranson
Sirius Inc.
P.O. Box 9258
760 Lynnhaven Parkway
Virginia Beach, VA 23452

Barbara Guyette
Marketing Specialist
Ada Products Division
Intermetrics, Inc.
733 Concord Av.
Cambridge, MA 02138

Sarah Hadley
National Security Agency
9800 Savage Road
Fort Meade, MD 20755-6000

Shmuel Halevi
Ad Cad Inc.
University Place, Suite 200
124 Mt. Auburn St.
Cambridge, MA 02138

Robert Haley
Director, SDI Programs
Cray Research, Inc.
1331 Pennsylvania Avenue, N.W.
Suite 1331 North
Washington, DC 2004

Margaret Hamilton, President
Hamilton Technologies, Inc.
17 Inman St.
Cambridge, MA 02139

Duane Harder
MS B-218
Los Alamos National Laboratory
Los Alamos, NM 87545

Evans C. Harrigan
Software Consultant
Cray Research, Inc.
2130 Main Street., Suite 280
Huntington Beach, CA 92648

Hal Hart
TRW Defense Systems Group
One Space Park
Redondo Beach, CA 90278

Goran Hemdahl
Technical Director
Advanced Systems Architectures
Johnson House, 73-79 Park Street
Camberley, Surrey GU15 3PE United Kingdom

Dale B. Henderson
Los Alamos National Laboratory
Receiving Department
Bldg. SM-30
Bikini Road
Los Alamos, NM 87545

John W. Hendricks
Systems Technologies, Inc.
242 Ocean Drive West
Stamford, CT 06902

Stephan L. Hise
Advanced Development Programs
Marketing Department - MS 1112
Westinghouse Electric Corporation
Defense Group
Friendship Site
Box 1693
Baltimore, MD 21203

Jung Pyo Hong
Los Alamos National Lab
MS K488
P.O. Box 1633
Los Alamos, NM 87544

Don Horne
SRS Technologies
1500 Wilson Blvd., Suite 800
Arlington, VA 22209-8707

Bill Horton
MITRE Corp.
1259 Lake Plaza Drive
Colorado Springs, CO 80906

Greg Janee
General Research Corp.
5383 Hollister Avenue
Santa Barbara, CA 93111

Andy Jazwinski, Director
Advanced Development
TASC - The Analytic Sciences Corporation
1700 N. Moore St., Suite 1220
Arlington, VA 22209

James R. Jill
Manager, Advanced Technologies
NTB Design
Martin Marietta Information & Communications Systems
P.O. Box 1260
Denver, CO 80201-1260

Sumalee Johnson
Rockwell Institute
P.O. Box 3644
Seal Beach, CA 90704-7644

W.G. (Gray) Jones
Science Applications International Corporation
1710 Goodridge Drive
McLean, VA 22102

Irene G. Kazakova
Director, Marketing
Interactive Development Environments
150 Fourth Street, Suite 210
San Francisco, CA 94103

Peter Keenan
Science Ltd.
Wavendon Towe
Milton, Keynes
England MK17-8LX

Judy Kerner
TRW R2/1134
One Space Park
Redondo Beach, CA 90278

Rebecca Kidd
General Research Corp.
307 Wynn Drive
Huntsville, AL 35805

Virginia P. Kobler
Chief, Technology Branch
Battle Management Division
Department of the Army
Office of the Chief of Staff
U.S. Army Strategic Defense Command
P.O. Box 1500
Huntsville, AL 35807-3801

Dr. Ijur Kulikov
Intermetrics
607 Louis Drive
Warminster, PA 18974

Lt. Ann Kuo
ESD/ATS
Hanscom AFB, MA 07831

John Michael Lake
2311 Galen Dr. #7
Champaign, IL 61821

John Latimer
Teledyne Brown Engineering
300 Sparkman Dr.
MS 44
Hunstville, AL 35807

Steve Layton
Senior Software Engineer
Martin Marietta Denver Aerospace
MS L0425
P.O. Box 179
Denver, CO 80201

Larry L. Lehman
Integrated Systems Inc.
2500 Mission College Road
Santa Clara, CA 95054

Eric Leighninger
Dynamics Research
60 Frontage Road
Andover, MA 01810

Peter Lempp
Software Products and Services, Inc.
14 East 38th Street, 14th Floor
New York, NY 10016

Bob Liley
Rockwell International Corporation
2600 West Minister Blvd.
Seal Beach, CA 90740-7644

Frank Poslajko
U.S. Army SDC
CSSD-H-SI
Huntsville, AL 35807-3801

Brian Smith
Mathematics & Computer Science Div.
Argonne National Laboratory
Building 221, Room C-219
9700 South Cass Avenue
Argonne, IL 60439-4844

Norman G. Snyder
Director of Software Services
Jodgrey Associates, Inc.
462 Highfield Ct.
Severna Park, MD 21146

J.R. Southern
USA-SDC
DASD-H-SBD
106 Wynn Drive
Huntsville, AL 35807-3801

Henry Sowizral
Schlumberger Palo Alto Research
3340 Hillview Avenue
Palo Alto, CA 94304

Stephen L. Squires
DARPA
Information Processing Techniques Office
1400 Wilson Blvd.
Arlington, VA 22209

C.E.R. Story
EASAMS Ltd.
Lyon Way, Frimley Road
Camberley, Surrey GU16 5EX

Dr. Richard D. Stutzke
Science Applications International Corporation
1710 Goodridge Dr.
McLean, VA 22102

Agapi Svolou
Senior Scientist
Manager of Software Science
Mellon Institute
Computer Engineering Center
4616 Henry St.
Pittsburgh, PA 15213-2683

Kathy Tammen
General Research Corp.
P.O. Box 6770
Santa Barbara, CA 93160-6770

Kenneth C. Taormina
Director, Anlaysis and Technology Requirements
Teledyne Brown Engineering
West Oaks Executive Park
3700 Pender Dr.
Fairfax, VA 22030

Edward Town
Rockwell International Corp.
2600 West Minister Blvd.
Seal Beach, CA 90740-7644

Larry Tubbs
US Army Strategic Defense Command
DASH-H-5B
106 Wynn Dr.
Huntsville, AL 35807

Charles M. Vairin
Martin Marietta Denver Aerospace
MS L8079
P.O. Box 179
700 W. Mineral Avenue
Littleton, CO 80201

Dr. Brooks Van Horne
TRW
One Federal Systems Park Drive
Fairfax, VA 22033

John L. Walsh
Riverside Research Institute
Washington Research Office
1701 North Fort Myer Drive, Suite 700
Arlington, VA 22209

G. Karl Warmbrod
SPARTA, Inc.
7926 Jones Branch Road
Suite 1070
Mclean, VA 22180

Erwin H. Warshawsky, President
JRS Research Laboratories, Inc.
202 W. Lincoln Av.
Orange, CA 92665-1040

Capt. Charles R. Waryk
OSD/SDIO/S/SA
Pentagon
Room 1E149
Washington, DC 20301-7100

Anthony Wasserman, President
Interactive Development Environments
150 Fourth St., Suite 210
San Francisco, CA 94103

Gio C. Weiderhold
Department of Computer Science
Stanford University
Stanford, CA 94305-2085

Bruce White
500 Montezuma Avenue, Suite 118
Santa Fe, NM 87501

Dave J. Whitley
Software Engineering Section
SCICON, Ltd.
Wavedon Tower
Wavedon Village
Milton Keynes, England MK178LX

John Wiley
BDM Corporation
2227 Drake Avenue
Huntsville, AL 83505

John D. Wolfe
Programmer/Analyst
Software Consulting Specialists, Inc.
P.O. Box 15367
Fort Wayne, IN 46885

Juan A. Wood
Los Alamos National Laboratory
Receiving Department
Bldg. SM-30
Bikini Road
Los Alamos, NM 87545

Richard M. Wright
0/96-01 B/30E
2100 East St. Elmo Road
Austin, TX 78744

Robert C. Yost
Corporate Vice-President
Director, Defense Research & Analysis Operation
SAIC (Science Applications International Corporation)
1710 Goodridge Dr.
McLean, VA 22102

Christine Youngblut
17021 Sioux Lane
Gaithersburg, MD 20878

Steve Zelazny
Science Applications International Corporation
4232 Ridge Lea Road
Amherst, NY 14226

Gerald A. Zionic
NTB Program Director
Martin Marietta Information & Communication Systems
P.O. Box 1260
Denver, CO 80201-1260

CSED Review Panel

Dr. Dan Alpert, Director — 1 copy
Center for Advanced Study
University of Illinois
912 W. Illinois Street
Urbana, Illinois 61801

Dr. Barry W. Boehm
TRW Defense Systems Group
MS 2-2304
One Space Park
Redondo Beach, CA 90278
1 copy

Dr. Ruth Davis
The Pymatuning Group, Inc.
2000 N. 15th Street, Suite 707
Arlington, VA 22201
1 copy

Dr. Larry E. Druffel
Software Engineering Institute
Shadyside Place
480 South Aiken Av.
Pittsburgh, PA 15231
1 copy

Dr. C.E. Hutchinson, Dean
Thayer School of Engineering
Dartmouth College
Hanover, NH 03755
1 copy

Mr. A.J. Jordano
Manager, Systems & Software
Engineering Headquarters
Federal Systems Division
6600 Rockledge Dr.
Bethesda, MD 20817
1 copy

Mr. Robert K. Lehto
Mainstay
302 Mill St.
Occoquan, VA 22125
1 copy

Mr. Oliver Selfridge
45 Percy Road
Lexington, MA 02173
1 copy

IDA

General W.Y. Smith, HQ	1 copy
Mr. Seymour Deitchman, HQ	1 copy
Mr. Philip Major, HQ	1 copy
Ms. Karen H. Weber, HQ	1 copy
Dr. Jack Kramer, CSED	1 copy
Dr. Robert I. Winner, CSED	1 copy
Dr. John Salasin, CSED	1 copy
Dr. Cathy J. Linn, CSED	100 copies
Dr. Joseph L. Linn, CSED	1 copy
Mr. Michael R. Kappel, CSED	1 copy
Mr. Howard Cohen, CSED	1 copy
Ms. Deborah Heystek, CSED	1 copy
Dr. Reginald Meeson, CSED	1 copy
Dr. Karen Gordon, CSED	1 copy
Dr. Norman Howes, CSED	1 copy
Dr. Dennis Fife, CSED	1 copy
Dr. Cy Ardoin, CSED	1 copy
Ms. Julia Sensiba, CSED	2 copies
Albert J. Perrella, Jr., STD	1 copy
IDA Control & Distribution Vault	3 copies

END

DATE

FILMED

9-88

DTIC